ROBUST CLUSTERING ALGORITHMS

A Thesis
Presented to
The Academic Faculty

by

Pramod Gupta

In Partial Fulfillment
of the Requirements for the Degree
Masters in Computer Science in the
School of Computer Science

Georgia Institute of Technology
May 2011

ROBUST CLUSTERING ALGORITHMS

Approved by:

Prof. Maria-Florina Balcan, Advisor
School of Computer Science
Georgia Institute of Technology

Prof. Santosh Vempala
School of Computer Science
Georgia Institute of Technology

Prof. Alexander Gray
School of Computer Science
Georgia Institute of Technology

Date Approved: March, 29, 2011

ACKNOWLEDGEMENTS

This thesis would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this work.

First and foremost, my utmost gratitude to Prof. Nina Balcan, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. She has been an immense inspiration and I thank her for her patience, motivation, enthusiasm, immense knowledge and steadfast encouragement as I hurdle all the obstacles in the completion this work.

I would like to thank the rest of my thesis committee: Prof. Santosh Vempala and Prof. Alex Gray for their encouragement and insightful comments throughout the course of this work.

I am extremely grateful to Prof. Dana Randall, an insightful and amazing teacher, who helped me better understand the theoretical thought process.

The Algorithms & Randomness Center, that allowed us to use their machines to run our experiments.

James Kriigel at the Technology Services Organization, College of Computing, for his help in setting us up with the necessary technical resources.

My friends and colleagues in the Theory Group and the School of Computer Science for some very insightful discussions and suggestions.

Lastly, I offer my sincerest gratitude to all of those who supported me in any respect during the completion of the project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Clustering is a method of *unsupervised learning* and a common technique for *statistical data analysis* where a data set is *segmented* into subsets such that the elements within each subset are somehow more *similar* to each other than to elements assigned to other subsets in some sense. Cluster analysis divides data into groups (clusters) that are either meaningful or useful or both. If meaningful clusters are the goal, then the clusters should capture the natural structure of the data. In some cases, however, cluster analysis is only a useful starting point for other purposes, such as data summarization.

Clustering techniques (Sokal & Michener, 1958; Gower, 1967; Anderberg, 1973; Sneath et al., 1973; Duda et al., 2000) have proved extremely useful in a wide range of application domains such as social sciences, biology, statistics, pattern recognition, image analysis, information retrieval, machine learning, data mining, etc. In recent times, these application domains have faced an explosion of data. As a consequence it has become increasingly important to develop effective, accurate, robust to noise, fast, and general clustering algorithms, accessible to developers and researchers in a diverse range of areas.

In this work, we propose two new agglomerative algorithms with theoretical guarantees for their robustness to noise. Further, we evaluate the strengths and weaknesses of the most popular algorithms through a systematic experimental analysis by comparing their performance and robustness to noise on a variety of synthetic and real-world data sets and show that both our algorithms consistently perform much better than other hierarchical algorithms and are much more robust to various forms of noise present in the data.

Many data clustering algorithms have been proposed in the literature (Johnson, 1967;

Hartigan & Wong, 1979; Jain & Dubes, 1981; Jain et al., 1999). Most of the clustering algorithms can be labeled as either hierarchical or partitional. A partitional algorithm divides a data set into a single partition, whereas a hierarchical algorithm divides a data set into a sequence of nested partitions. Hierarchical clustering methods are one of the oldest and most commonly used clustering algorithms.

In hierarchical clustering (Johnson, 1967; Duda et al., 2000; Dasgupta & Long, 2005), the goal is to find a hierarchy (generally represented by a tree) of partitions of data which may reveal interesting structure in the data at multiple levels of granularity. Hierarchical clustering algorithms are either top-down (divisive) (Dasgupta et al., 2006) or bottom-up (agglomerative) (Sneath et al., 1973; KING, 1967; Gower, 1967). *Bottom-up clustering* methods treat each point as a singleton cluster at the outset and then successively merge (or agglomerate) pairs of clusters until all clusters have been merged into a single cluster that contains all points. *Top-down clustering* proceeds by splitting clusters recursively until individual points are reached. Agglomerative hierarchical clustering techniques are by far the most common, and the main focus of this work.

In all agglomerative algorithms a measure of similarity between pairs of points is assumed and different schemes are distinguished by how they convert this similarity information into a measure of similarity between two clusters containing these points. For example, in *single linkage* (Sneath et al., 1973) the similarity between two clusters is the maximum similarity between points in these two different clusters whereas in *complete linkage* (KING, 1967) the similarity between two clusters is the minimum similarity between points in these two different clusters.

Agglomerative algorithms are extremely popular in a wide range of application domains ranging from biology applications to social sciences to computer vision applications mainly because they are fast and their output is easy to interpret. Moreover, any valid measure of similarity can be used, in fact, the observations themselves are not required if

we have a valid matrix of pairwise similarities. The number of clusters need not be specified beforehand and problems due to incorrect initialization and local minima do not arise (Frigui & Krishnapuram, 1997).

However, one of the main limitations of these methods is that they are not robust to noise or outliers (Narasimhan et al., 2005; Balcan & Gupta, 2010a). Since they consider only local neighbors at each step, they cannot incorporate any *a priori* knowledge about the global shape or size of clusters. Moreover, the assignments are static, and points committed to a given cluster in the early stages cannot be moved to a different cluster afterwards. As a result, agglomerative algorithms do not perform well when clusters have noisy points that are similar to points in other clusters.

Several algorithms have been proposed to remedy these limitations and agglomerative clustering techniques have experienced rapid improvements in many aspects. (Wishart, 1969) proposed a new robust algorithm to overcome the insensitivity of single linkage to data density which causes *chaining* (low density noisy points tying larger cohesive clusters together). At each step, Wishart's method avoids chaining by discarding as noise regions with density lower than a particular threshold before merging clusters, ensuring regions of high density become available for linkage earlier than regions of low density. However, it was unclear how to choose the input parameter for the algorithm till (Chaudhuri & Dasgupta, 2010) proposed a generalization of Wishart's method and gave a theoretical basis for choosing the input parameter. Unfortunately, both these methods are not robust to *sparse noise* (a large fraction of the points in the data have very high similarity to a few points from other target clusters even though most of the nearest neighbors are from their own target cluster). (Guha et al., 1998) presented a new algorithm, *CURE*, that is robust to outliers and capable of clustering data of any shape using a combination of random sampling and partition clustering to handle large databases but cannot handle data where clusters have differing densities. Like Wishart's methods, *CURE* can also be shown to fail in case of sparse noise. *BIRCH* was proposed by (Zhang et al., 1996) as a *hybrid clustering* method

(Murty & Krishna, 1981) that first partitions the input data into sub-clusters and then constructs a hierarchy based on these sub-clusters. It fails when clusters do not have uniform sizes and shapes (convex or spherical) and is only applicable for similarity measures that are metric and cannot be used in general. Moreover, Wishart's method, *CURE* and *BIRCH* lacked any theoretical guarantees for their correctness.

This brings up the question: *is it possible to design effective, fast and generic agglomerative clustering algorithms that have formal guarantees for their robustness and can tolerate a substantial degree of noise*?

The contribution of this work is to provide a strong positive answer to this question; we develop two new robust, linkage based algorithms, *Robust Hierarchical Linkage (RHL)* and *Weighted Neighborhood Linkage (WNL)*, for agglomerative hierarchical clustering, with formal guarantees for their robustness, that will succeed in many interesting cases where standard agglomerative algorithms will fail. At a high level, *RHL* is robust to noise in two different ways. First, it uses more global information for creating an interesting starting point for a linkage procedure; second, it uses a robust linkage procedure for merging large enough blobs. Similarly, *WNL* too uses global structural properties of the data while merging clusters which makes it robust.

In order to formally analyze correctness of our algorithm we use the framework introduced by (Balcan et al., 2008). In this framework, we assume there is some target clustering (much like a k-class target function in the multi-class learning setting) and we say that an algorithm correctly clusters data satisfying property P if on any data set having property P, the algorithm produces a tree such that the target is some pruning of the tree. For example if all points are more similar to points in their own cluster than to points in any other cluster (this is called the strict separation property), then any of the standard agglomerative algorithms will succeed. However, with just tiny bit of noise, for example if each point has even just one point from a different cluster that it is similar too, then the standard algorithms will all fail.

Under this framework, we propose two new natural properties of data in which all points in the data set are allowed to be affected by some amount of noise called the *good neighborhood* properties. These properties capture extremely well various forms of real world noise like presence of outliers, incompleteness of data, data corruption, transmission errors, etc. The good neighborhood property roughly says that after a small number of extremely malicious points have been removed, for the remaining points in the data set, most of their nearest neighbors are from their target cluster.

We show that if the data satisfies these *good neighborhood* properties, then our algorithms can be used to cluster well in the tree model (*i.e.* to output a hierarchy such that the target clustering is a pruning of that hierarchy). In particular, we can show that if the data satisfies the (α, ν)-good neighborhood property, then RHL will be successful in generating a hierarchy such that the target clustering is a pruning of that hierarchy, whereas WNL will be successful if the data satisfies the $(\alpha, 0)$-good neighborhood property. We note here that the *good neighborhood* properties are insensitive to any monotone transformation of the underlying similarity functions and therefore our algorithms, RHL and WNL too are insensitive to any monotone transformation of the similarity functions.

We also show how to extend our algorithms to an inductive setting with similar correctness guarantees for robustness. In the inductive setting, first a small subset of points is randomly chosen from a much larger instance space to generate a hierarchy into which the rest of the points are then inserted resulting in a hierarchy over the entire instance space. This contrasts sharply with the fact that there are no known ways of extending the standard linkage algorithms to the inductive setting and thus have to be run over the entire data set. Moreover, the inductive versions of our algorithms require only a small random sample which is independent on the size of instance space and depends only on the noise parameters α, ν and the confidence parameter δ.

We then do a systematic experimental analysis of several popular hierarchical clustering algorithms like *Standard Linkage algorithms, (Generalized)Wishart's Method, CURE,*

EigenCluster, etc. and compare their results with our algorithms *RHL* and *WNL* on a wide variety of synthetic and real-world data sets (Frank & Asuncion, 2010) and show that both our algorithms consistently perform much better than other hierarchical algorithms and are much more robust to various forms of noise present in the data. Further, we show experimentally the efficacy of the inductive versions of our algorithms as a quicker and simpler alternative when analysis over the complete data set is difficult or even prohibitive.

We note here that while being a robust algorithm, *RHL* is not completely agglomerative and is algorithmically quite complex and computationally intensive due to it's running time of $O(n^{\omega+1})$. Moreover, it requires two parameters α, ν as input. On the other hand, though, *WNL* can be proven theoretically only for the simpler $(\alpha, 0)$-good neighborhood property, it turns out to be a more practical algorithm as compared to *RHL* as it is provably faster, algorithmically simpler, completely agglomerative in nature, requires only a single input parameter and can be shown experimentally to perform comparably to *RHL* and to be much more robust to parameter tuning, *i.e.* incorrect estimates of the input parameter α.

Summary of Main Results and Bibliographic Information

This thesis is organized as follows:

- In Chapter 3 we describe the framework used for theoretical analysis of algorithms and within this framework, propose two new natural properties of data in which all points in the data set are allowed to be affected by some amount of noise called the *good neighborhood* properties. This is based on the work that appears in (Balcan & Gupta, 2010a).

- In Chapter 4 we present the *Robust Hierarchical Linkage (RHL)* algorithm and give theoretical guarantees for it's robustness when the data is noisy, *i.e.* if the data satisfies the good neighborhood property. In particular, we can show that if the data satisfies the (α, ν)-good neighborhood property, then *RHL* will be successful in generating a hierarchy such that the target clustering is a pruning of that hierarchy. This

chapter is based on the work that appears in (Balcan & Gupta, 2010a).

- In Chapter 5 we present the *Weighted Neighborhood Linkage* algorithm and give theoretical guarantees for it's robustness when the data is noisy, *i.e.* if the data satisfies the good neighborhood property. In particular, we can show that if the data satisfies the $(\alpha, 0)$-good neighborhood property, then WNL will be successful in generating a hierarchy such that the target clustering is a pruning of that hierarchy. This chapter is based on the work that appears in (Balcan & Gupta, 2010b).

- In Chapter 6 we extend both the algorithms, RHL and WNL to an inductive setting and give theoretical guarantees for their robustness in this setting. This chapter is based on the work that appears in (Balcan & Gupta, 2010a,b).

- In Chapter 7 we do a systematic experimental analysis of several popular hierarchical clustering algorithms and compare their results with RHL and WNL on a wide variety of synthetic and real-world data sets and show that both our algorithms consistently perform much better than other hierarchical algorithms and are much more robust to various forms of noise present in the data. Further, we show experimentally the efficacy of the inductive versions of our algorithms as a quicker and simpler alternative when analysis over the complete data set is difficult or even prohibitive. This chapter is based on the work that appears in (Balcan & Gupta, 2010b).

CHAPTER I

CLUSTERING ANALYSIS: AN INTRODUCTION

In this chapter we introduce clustering analysis and give a formal definition of what a clustering of data set means.

In Section 1.3, we discuss and compare different types of clusterings such as *hierarchical & partitional*, *hard & fuzzy*, *agglomerative & divisive*, etc.

In Section 1.4, we introduce different types an attribute in the data set can have. Understanding data types is extremely important as all clustering algorithms are very much associated with data types. Therefore, understanding scale, normalization, and similarity is very important in interpreting the results of clustering algorithms.

All clustering algorithms assume some measure of similarity between pairs of points and different schemes are distinguished by how they convert this similarity information into a measure of similarity between two clusters containing these points while forming clusters, performing merging or splitting operations, etc. In Section 1.5, we discuss some widely used similarity and dissimilarity measures between data points like *covariance matrix, euclidean distance, minkowski distance, cosine similarity*, etc. The more the two data points resemble one another, the larger the similarity coefficient is.

1.1 Introduction

As the amount of data we nowadays have to deal with becomes larger and larger, the methods that help us to detect structures in the data and to identify interesting subsets in the data become more and more important. One of these methods is clustering, *i.e. segmenting* a set into subsets such that the elements in each subset are somehow *similar* to each other and elements of different subsets are *dissimilar*.

Cluster analysis (Sokal & Michener, 1958; Johnson, 1967; Gower, 1967; Sneath et al.,

1973; Duda et al., 2000) divides data into groups (clusters) that are either meaningful or useful or both. If meaningful clusters are the goal, then the clusters should capture the natural structure of the data. In some cases, however, cluster analysis is only a useful starting point for other purposes, such as data summarization. Whether for understanding or utility, cluster analysis has long played an important role in a wide variety of fields: psychology and other social sciences, biology, statistics, pattern recognition, information retrieval, machine learning, and data mining. There have been many applications of cluster analysis to practical problems.

Clustering for Understanding: Classes, or conceptually meaningful groups of objects that share common characteristics, play an important role in how people analyze and describe the world. Indeed, human beings are skilled at dividing objects into groups (clustering) and assigning particular objects to these groups (classification). For example, even relatively young children can quickly label the objects in a photograph as buildings, vehicles, people, animals, plants, etc. In the context of understanding data, clusters are potential classes and cluster analysis is the study of techniques for automatically finding classes. In Biology, clustering has been used to find groups of genes that have similar functions. In Information Retrieval, clustering can be used to group search results of a query into a small number of clusters, each of which capturing a particular aspect of the query. In Geology, cluster analysis has been applied to find patterns in the atmospheric pressure of polar regions and areas of the ocean that have a significant impact on land climate. In Medicine, cluster analysis can also be used to detect patterns in the spatial or temporal distribution of diseases like cancer, autism, etc.

Clustering for Utility: Cluster analysis provides an abstraction from individual data objects to the clusters in which those data objects reside. Additionally, some clustering techniques characterize each cluster in terms of a cluster prototype; *i.e.* a data object that is representative of the other objects in the cluster. These cluster prototypes can be used as the

basis for a number of data analysis or data processing techniques. Therefore, in the context of utility, cluster analysis is the study of techniques for finding the most representative cluster prototypes. For example, instead of applying data analysis techniques, having high space and time complexities, to the entire data set, it is much more efficient to apply them to a reduced data set consisting only of prototypes generated through clustering. Cluster prototypes can also be used for data compression where each object in the data set is represented by the index of the prototype associated with it's cluster. This type of compression is known as *vector quantization* and is often applied to image, sound, and video data.

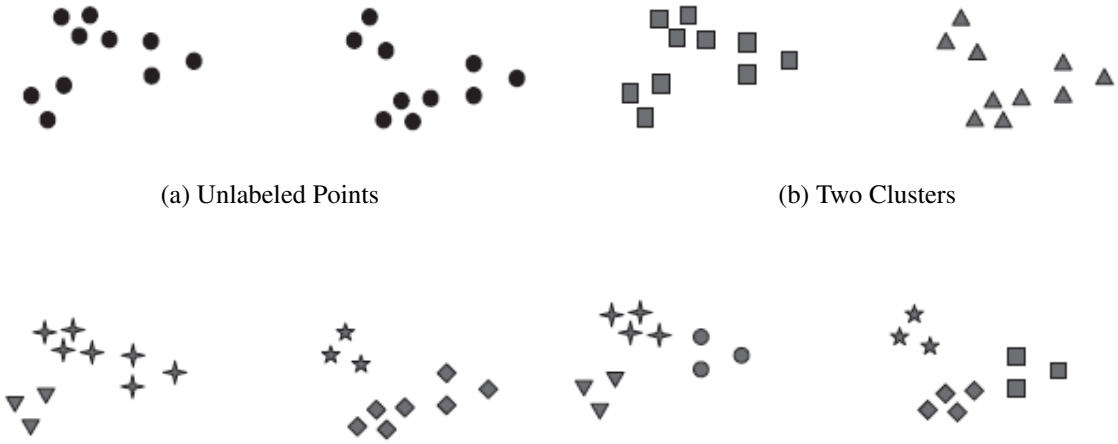
Cluster analysis groups data objects based only on information found in the data that describes the objects and their relationships. The goal is that the objects within a group be similar (or related) to one another and different from (or unrelated to) the objects in other groups. The greater the similarity (or homogeneity) within a group and the greater the difference between groups, the better or more distinct the clustering.

In many applications, the notion of a cluster is not well defined. To better understand the difficulty of deciding what constitutes a cluster, consider Figure 1a, which shows twenty points and three different ways of dividing them into clusters. The shapes of the markers indicate cluster membership. Figures 1b and 1d divide the data into two and six parts respectively. However, the apparent division of each of the two larger clusters into three subclusters may simply be an artifact of the human visual system. Also, it may not be unreasonable to say that the points form four clusters, as shown in Figure 1c. This figure illustrates that the definition of a cluster is imprecise and that the best definition depends on the nature of data and the desired results.

1.2 Formal Definition

Given a *data set* S containing n points, a clustering C is a partition of S into K non-empty, disjoint sets $\{C_1, C_2, \ldots, C_K\}$ called *clusters* such that

$$C_i \cap C_j = \phi \quad and \quad \bigcup_{i=1}^{k} C_i = S \tag{1}$$



(a) Unlabeled Points

(b) Two Clusters

(c) Four Clusters

(d) Six Clusters

Figure 1: Different clusterings of the same data set.

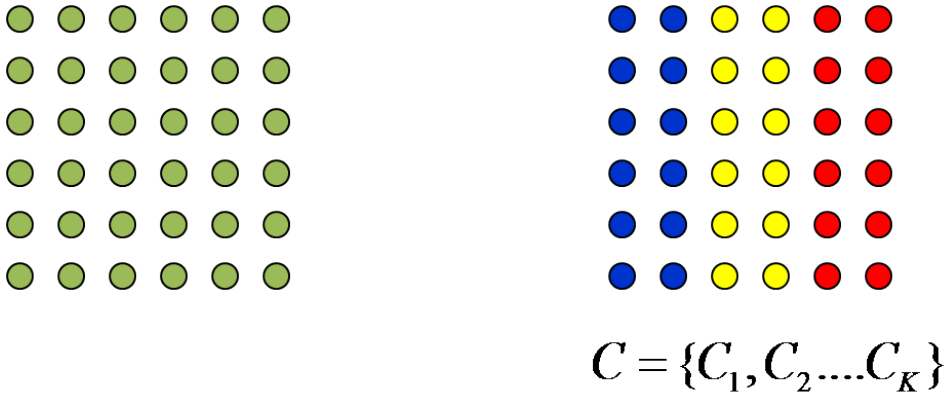


Figure 2: A clustering

The number of points in the cluster C_i is denoted by n_i such that $n_i \geq 1$ and

$$n = \sum_{i=1}^{K} n_i$$

1.3 Types of Clusterings

An entire collection of clusters is commonly referred to as a **clustering**. In this section, we distinguish various types of clusterings.

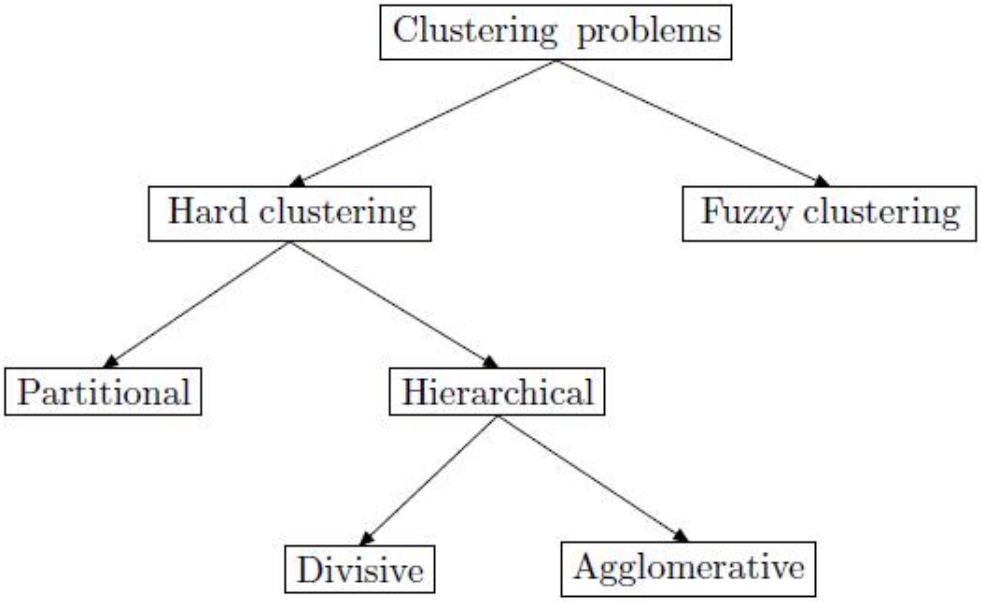


Figure 3: Types of Clusterings

1.3.1 Hierarchical vs Partitional

The most commonly discussed distinction among different types of clusterings is whether the set of clusters is nested or unnested, or in more traditional terminology, hierarchical or partitional. A partitional clustering is simply a division of the set of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset. Taken individually, each collection of clusters in Figures 1b - 1d is a partitional clustering.

If we permit clusters to have subclusters, then we obtain a hierarchical clustering, which is a set of nested clusters that are organized as a tree. Each node (cluster) in the tree (except for the leaf nodes) is the union of it's children (subclusters), and the root of the tree is the cluster containing all the objects. Often, but not always, the leaves of the tree are singleton clusters of individual data objects. If we allow clusters to be nested, then one interpretation of Figure 8.1(a) is that it has two subclusters (Figure 1b), each of which, in turn, has three subclusters (Figure 1d). The clusterings shown in Figure 1, when taken in that order, also form a hierarchical (nested) clustering with, respectively, $1, 2, 4,$ *and* 6 clusters on each level. Finally, note that a hierarchical clustering can be viewed as a sequence of

partitional clusterings and a partitional clustering can be obtained by taking any member of that sequence; *i.e.* by cutting the hierarchical tree at a particular level.

1.3.2 Hard (Exclusive or Overlapping) vs Fuzzy

The clusterings shown in Figure 1 are all **exclusive**, as they assign each object to a single cluster. There are many situations in which a point could reasonably be placed in more than one cluster, and these situations are better addressed by non-exclusive clustering.

In the most general sense, an **overlapping** or non-exclusive clustering is used to reflect the fact that an object can simultaneously belong to more than one group (class). For instance, a person at a university can be both an enrolled student and an employee of the university. A non-exclusive clustering is also often used when, for example, an object is *between* two or more clusters and could reasonably be assigned to any of these clusters. Imagine a point halfway between two of the clusters of Figure 1. Rather than make a somewhat arbitrary assignment of the object to a single cluster, it is placed in all of the *equally good* clusters.

In **fuzzy clustering**, every object belongs to every cluster with a membership weight that is between 0 (absolutely doesn't belong) and 1 (absolutely belongs). In other words, clusters are treated as fuzzy sets. Mathematically, a fuzzy set is one in which an object belongs to any set with a weight that is between 0 and 1. In fuzzy clustering, we often impose the additional constraint that the sum of the weights for each object must equal 1.) Similarly, probabilistic clustering techniques compute the probability with which each point belongs to each cluster, and these probabilities must also sum to 1. Because the membership weights or probabilities for any object sum to 1, a fuzzy or probabilistic clustering does not address true multiclass situations, such as the case of a student employee, where an object belongs to multiple classes. Instead, these approaches are most appropriate for avoiding the arbitrariness of assigning an object to only one cluster when it may be close

to several. In practice, a fuzzy or probabilistic clustering is often converted to an exclusive clustering by assigning each object to the cluster in which it's membership weight or probability is highest.

1.3.3 Complete vs Partial

A **complete clustering** assigns every object to a cluster, whereas a **partial clustering** does not. The motivation for a partial clustering is that some objects in a data set may not belong to well-defined groups. Many times objects in the data set may represent noise, outliers, or uninteresting background. For example, some newspaper stories may share a common theme, such as global warming, while other stories are more generic or one-of-a-kind. Thus, to find the important topics in last month's stories, we may want to search only for clusters of documents that are tightly related by a common theme. In other cases, a complete clustering of the objects is desired. For example, an application that uses clustering to organize documents for browsing needs to guarantee that all documents can be browsed.

1.3.4 Agglomerative vs Divisive

This aspect relates to algorithmic structure and operation. An **agglomerative** approach begins with each pattern in a distinct (singleton) cluster, and successively merges clusters together until a stopping criterion is satisfied. A **divisive** method begins with all patterns in a single cluster and performs splitting until a stopping criterion is met.

1.3.5 Deterministic vs Stochastic

This is most relevant to partitional approaches designed to optimize a squared error function. This optimization can be accomplished using traditional techniques or through a random search of the state space consisting of all possible labelings.

1.3.6 Incremental vs Non-Incremental

This issue arises when the data set to be clustered is large, and constraints on execution time or memory space affect the architecture of the algorithm. The early history of clustering methodology does not contain many examples of clustering algorithms designed to work with large data sets, but the advent of data mining has fostered the development of clustering algorithms that minimize the number of scans through the data set, reduce the number of objects examined during execution, or reduce the size of data structures used in the algorithm's operations.

1.4 Types of Data

Clustering algorithms are very much associated with data types. Therefore, understanding scale, normalization, and similarity is very important in interpreting the results of clustering algorithms. Data type refers to the degree of quantization in the data (Jain & Dubes, 1981) a single attribute can be typed as binary, discrete, or continuous. A binary attribute has exactly two values, such as true or false. A discrete attribute has a finite number of possible values, thus binary types are a special case of discrete types.

Below we discuss some basic types of data encountered in cluster analysis. In the real world, however, there exist various other data types, such as image data, spatial data, etc. Also, a data set may consist of multiple data types, such as a data set containing categorical data and numerical data. To conduct cluster analysis on data sets that contain unusual types of data, the similarity or dissimilarity measures should be defined in a meaningful way. The types of data are highly area specific, for example, DNA data in biology or financial time series in marketing research.

1.4.1 Numerical Data

A numerical attribute is an attribute that can take any value within a defined range. This is the most common type of attribute found in most data sets. A numerical attribute can

be either discrete or continuous. It is discrete if the variable is defined on a finite set of possible values and continuous if it can take any possible value within the predefined range. For example, attributes of person such as height, weight age, etc. are numerical attributes.

1.4.2 Binary Data

A binary attribute is an attribute that has exactly two possible values, such as *true* or *false*. Note that binary variables can be further divided into two types: symmetric binary variables and asymmetric binary variables. In a symmetric binary variable, the two values are equally important. An example is male-female. Symmetric binary variables are nominal variables. In an asymmetric variable, one of it's values carries more importance than the other. For example, yes stands for the presence of a certain attribute and no stands for the absence of a certain attribute.

1.4.3 Categorical Data

Categorical attributes are also referred to as nominal attributes, which are simply used as names, such as the brands of cars and names of bank branches. Since we consider data sets with a finite number of data points, a nominal attribute of the data points in the data set can have only a finite number of values; thus the nominal type is also a special case of the discrete type.

1.4.4 Transaction Data

Given a set of items I, a transaction is a subset of I. A transaction data set S is a set of transactions. Transactions can be represented by binary vectors, in which each entry denotes the presence or absence of the corresponding item. From this point of view, transaction data are a special case of binary data. The most common example of transaction data is market basket data. In a market basket data set, a transaction contains a subset of the total set of items that could be purchased. Generally, many transactions are made of sparsely distributed items. For example, a customer may only buy several items from a store with

thousands of items.

1.4.5 Time Series Data

Time series are the simplest form of temporal data. Precisely, a time series is a sequence of real numbers that represent the measurements of a real variable at equal time intervals. For example, stock price movements, the temperature at a given place, and volume of sales over time are all time series. A time series is discrete if the variable is defined on a finite set of time points. Most of the time series encountered in cluster analysis are discrete time series. When a variable is defined at all points in time, then the time series is continuous.

1.5 Similarity Measure

This section introduces some widely used similarity and dissimilarity measures between data points. A similarity coefficient indicates the strength of the relationship between two data points (Everitt, 1993). The more the two data points resemble one another, the larger the similarity coefficient is. Let $x = (x_1, x_2, ..., x_d)$ and $y = (y_1, y_2, ..., y_d)$ be two d-dimensional data points. Then the similarity coefficient between x and y will be some function of their attribute values, *i.e.*

$$\mathcal{K}(x, y) = \mathcal{K}(x_1, x_2, ..., x_d, y_1, y_2, ..., y_d) \tag{2}$$

A similarity function (Balcan et al., 2008) $\mathcal{K}$ usually lies in the range $[-1, 1]$. A point has a similarity of 1 to itself, *i.e.* $\mathcal{K}(x, x) = 1$. $\mathcal{K}$ can either be symmetric, *i.e.* $\mathcal{K}(x, y) = \mathcal{K}(y, x)$ or asymmetric *i.e.* $\mathcal{K}(x, y) \neq \mathcal{K}(y, x)$ (Gower, 1967).

A distance function f may be metric. A **metric** is a distance function f that satisfies the following four properties (Anderberg, 1973):

1. nonnegativity: $f(x, y) \geq 0$
2. reflexivity: $f(x, y) = 0 \Leftrightarrow x = y$
3. commutativity: $f(x, y) = f(y, x)$

4. triangle inequality: $f(x, y) \leq f(x, z) + f(z, y)$

where x, y, z are arbitrary data points.

The choice of distances is important for applications, and the best choice is often achieved via a combination of experience, skill, knowledge, and luck. Here we list some commonly used distances.

1.5.1 Covariance Matrix

Covariance is a well-known concept in statistics. Let S be a data set with n objects, each of which is described by d attributes $v_1, v_2, ..., v_d$. The attributes $v_1, v_2, ..., v_d$ are also referred to as variables. The covariance between two variables v_r and v_s is defined to be the ratio of the sum of the products of their deviation from the mean to the number of objects (Rummel, 1970), *i.e.*

$$c_{rs} = \frac{1}{n} \sum_{i=1}^{n} (x_{ir} - \bar{x}_r)(x_{is} - \bar{x}_s)$$

where x_{ij} is the j_{th} component of data point x_i and $\bar{x}_j$ is the mean of all data points in the j_{th} variable, *i.e.*

$$\bar{x}_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij}$$

The covariance matrix is a $d \times d$ matrix in which entry (r, s) contains the covariance between variable v_r and v_s , *i.e.*

$$\Sigma = \begin{pmatrix} c_{11} & c_{12} & \cdot & \cdot & \cdot & c_{1d} \\ c_{21} & c_{22} & \cdot & \cdot & \cdot & c_{2d} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ c_{d1} & c_{d1} & \cdot & \cdot & \cdot & c_{dd} \end{pmatrix} \tag{3}$$

From the definition of the covariance, the covariance matrix defined in Eq. (3) can be written as

$$\Sigma = \frac{1}{n} \mathcal{X}^T \mathcal{X}$$

where X^T denotes the transpose of $\mathcal{X}$, and $\mathcal{X}$ is a $d \times d$ matrix with the $(i, j)_{th}$ element $x_{ij} - \bar{x}_j$, *i.e.*

$$\mathcal{X} = (x_{ij} - \bar{x}_j)_{n \times d} = \begin{pmatrix} x_1 - \bar{x}_1 I_d \\ x_2 - \bar{x}_2 I_d \\ . \\ . \\ . \\ x_1 - \bar{x}_1 I_d \end{pmatrix} \tag{4}$$

where I_d is the d-dimensional identity vector, *i.e.* $I_d = (1, 1, ..., 1)$.

1.5.2 Euclidean Distance

Euclidean distance is probably the most common distance we have ever used for numerical data. For two data points x and y in d-dimensional space, the Euclidean distance between them is defined to be

$$d_{euc}(x, y) = \left(\sum_{i=1}^{d} (x_i - y_i)^2 \right)^{\frac{1}{2}} \tag{5}$$

where x_i and y_i are the values of the j_{th} feature of x and y respectively.

The **Squared Euclidean Distance** is defined to be

$$d_{seuc}(x, y) = d_{seuc}(x, y)^2 = \left(\sum_{i=1}^{d} (x_i - y_i)^2 \right) \tag{6}$$

1.5.2.1 Average Distance

The Euclidean distance has a drawback that two data points with no attribute values in common may have a smaller distance than another pair of data points containing the same attribute values. In this case, the average distance is a better alternative (Legendre & Legendre, 1983). The average distance is modified from the Euclidean distance. Given two data points x and y in d-dimensional space, the average distance is defined by

$$d_{ave}(x, y) = \left(\frac{1}{d} \sum_{i=1}^{d} (x_i - y_i)^2 \right)^{\frac{1}{2}} \tag{7}$$

1.5.3 Manhattan Distance

Manhattan distance is also called city block distance and is defined to be the sum of the distances of all attributes. That is, for two data points x and y in d-dimensional space, the Manhattan distance between them is

$$d_{man}(x, y) = \sum_{i=1}^{d} |(x_i - y_i)| \tag{8}$$

If the data point x or y has missing values at some attributes, then the Manhattan distance can be defined as (Wishart, 2003)

$$d_{manw}(x, y) = \sum_{i=1}^{d} \frac{w_i|(x_i - y_i)|}{\sum_{i=1}^{d} w_i} \tag{9}$$

where $w_i = 1$ if both x and y have observations of the i_{th} attribute and 0 otherwise.

The **Manhattan Segmental Distance** is a variant of the Manhattan distance. In this, only a part of the whole dimension is used to calculate the distance. It is defined as (Aggarwal et al., 1999)

$$d_P(x, y) = \sum_{i \in P} \frac{|(x_i - y_i)|}{|P|} \tag{10}$$

where P is a nonempty subset of $\{1, 2, ..., d\}$.

1.5.4 Maximum Distance

The maximum distance is also called the *sup* distance. It is defined to be the maximum value of the distances of the attributes; that is, for two data points x and y in d-dimensional space, the maximum distance between them is

$$d_{max}(x, y) = \max_{1 \le i \le d} |(x_i - y_i)| \tag{11}$$

1.5.5 Minkowski Distance

The Euclidean distance, Manhattan distance, and maximum distance are three particular cases of the Minkowski distance defined by

$$d_{min}(x, y) = \left(\sum_{i=1}^{d} (x_i - y_i)^r \right)^{\frac{1}{r}}, \quad r \ge 1 \tag{12}$$

where r is called the order of the Minkowski distance.

Note: If we take $r = 2, 1, \ and \ \inf$, we get the Euclidean distance, Manhattan distance, and maximum distance, respectively.

If the data set has compact or isolated clusters, the Minkowski distance works well, otherwise the largest-scale attribute tends to dominate the others. To avoid this, we should normalize the attributes or use weighting schemes (Jain et al., 1999).

1.5.6 Mahalanobis Distance

Mahalanobis distance (Jain & Dubes, 1981) can alleviate the distance distortion caused by linear combinations of attributes. It is defined by

$$d_{mah}(x, y) = \sqrt{(x - y)\Sigma^{-1}(x - y)^T} \tag{13}$$

where Σ is the covariance matrix of the data set defined in Eq. (3). Therefore, this distance applies a weight scheme to the data.

Another important property of the Mahalanobis distance is that it is invariant under all nonsingular transformations. For example, let C be any nonsingular $d \times d$ matrix applied to the original data set $S = \{x_1, x_2, ..., x_n\}$ by

$$y_i = Cx_i, \qquad i = 1 \ to \ n$$

The new Covariance matrix becomes

$$\frac{1}{n}Y^T Y = \frac{1}{n}(XC^T)^T(XC^T)$$

where X is defined in Eq. (4) and Y is defined similarly for the transformed data set. Then

the Mahalanobis distance between y_i and y_j is

$$\begin{aligned} d_{mah}(y_i, y_j) &= \sqrt{(y_i - y_j)\left(\frac{1}{n}Y^T Y\right)^{-1}(y_i - y_j)^T} \\ &= \sqrt{(x_i - x_j)C^T\left(\frac{1}{n}(XC^T)^T(XC^T)\right)^{-1}C(x_i - x_j)^T} \\ &= \sqrt{(x_i - x_j)\left(\frac{1}{n}X^T X\right)^{-1}(x_i - x_j)^T} \\ &= d_{mah}(x_i, x_j) \end{aligned}$$

which shows that the Mahalanobis distance is invariant under nonsingular transformations.

The Mahalanobis distance suffers from some disadvantages. For example, it involves high computation, since the covariance matrix is computed based on all data points in the data set.

1.5.7 Cosine Similarity

The cosine similarity measure (Salton & McGill, 1983) is adopted to measure the similarity between data. Let x and y denote two d-dimensional data points. Then the cosine similarity between x and y is given by

$$\mathcal{K}_{cos}(x, y) = \frac{x \cdot y}{||x||||y||} \tag{14}$$

For *text matching*, the attribute vectors x and y are usually the term frequency vectors of the documents. The cosine similarity can be seen as a method of normalizing document length during comparison. The resulting similarity ranges from -1 meaning exactly opposite, to 1 meaning exactly the same, with 0 usually indicating independence, and in-between values indicating intermediate similarity or dissimilarity.

In the case of *information retrieval*, the cosine similarity of two documents will range from 0 to 1, since the term frequencies ($tf - idf$ weights) cannot be negative. The angle between two term frequency vectors cannot be greater than 90.

1.5.8 A General Similarity Coefficient

The general similarity coefficient (Gower, 1971) has been widely implemented and used to measure the similarity for two mixed-type data points. This general similarity coefficient can also be applied to data points with missing values. Let x and y denote two d-dimensional data points. Then the general similarity measure $\mathcal{K}_{gower}(x, y)$ is defined as

$$\mathcal{K}_{gower}(x, y) = \frac{\sum_{i=1} dw(x_i, y_i)\mathcal{K}(x_i, y_i)}{\sum_{i=1} dw(x_i, y_i)} \tag{15}$$

where $\mathcal{K}(x_i, y_i)$ is a similarity component for the i_{th} attribute and $w(x_i, y_i)$ is either 1 or 0 depending on whether or not a comparison is valid for the i_{th} attribute of the two data points. They are defined respectively for different attribute types.

- For *quantitative attributes* x_i and y_i, $\mathcal{K}(x_i, y_i)$ is defined as

  $$\mathcal{K}(x_i, y_i) = 1 - \frac{|x_i - y_i|}{R_i}$$

  where R_i is the range of the i_{th} attribute. $w(x_i, y_i) = 0$ if data point x or y has missing value at the i_{th} attribute; otherwise $w(x_i, y_i) = 1$.

- For *quantitative attributes* x_i and y_i, $\mathcal{K}(x_i, y_i) = 1$ if both data points x and y have the i_{th} attribute present and 0 otherwise. $w(x_i, y_i) = 0$ if both data points x and y have missing value at the i_{th} attribute; otherwise $w(x_i, y_i) = 1$.

- For *nominal or categorical attributes* x_i and y_i, $\mathcal{K}(x_i, y_i) = 1$ if $x = y$ and 0 otherwise. $w(x_i, y_i) = 0$ if data point x or y has missing value at the i_{th} attribute; otherwise $w(x_i, y_i) = 1$.

From the definition of the general similarity coefficient, we see that $\mathcal{K}_{gower}(x, y)$ achieves maximum value 1 if the two data points are identical and has minimum value 0 if the two data points are extremely different.

1.6 Discussion

In this chapter we introduced clustering analysis and gave a formal definition of what a clustering of data set means. Then we discussed and compared different types of clusterings such as *hierarchical & partitional*, *hard & fuzzy*, *agglomerative & divisive*, etc. We note here that *hierarchical clustering* is the focus of this work and it is discussed in significant detain in the ensuing chapters.

We introduced different types an attribute in the data set can have. Understanding data types is extremely important as all clustering algorithms are very much associated with data types. Therefore, understanding scale, normalization, and similarity is very important in interpreting the results of clustering algorithms. However, we only discussed some basic types of data encountered in cluster analysis. In the real world, however, there exist various other data types, such as image data, spatial data, etc. Also, a data set may consist of multiple data types, *e.g.* a data set containing categorical data and numerical data. To conduct cluster analysis on data sets that contain unusual types of data, the similarity or dissimilarity measures should be defined in a meaningful way.

Then we discussed some widely used similarity and dissimilarity measures between data points like *covariance matrix, euclidean distance, minkowski distance, cosine similarity*, etc. The more the two data points resemble one another, the larger the similarity coefficient is. All clustering algorithms assume some measure of similarity between pairs of points and different schemes are distinguished by how they convert this similarity information into a measure of similarity between two clusters containing these points while forming clusters, performing merging or splitting operations, etc. We note here that using the right similarity measure for a data set is extremely important as using different similarity measures may result in completely different clusterings of the same data set.

CHAPTER II

HIERARCHICAL CLUSTERING

Clustering algorithms are subdivided into hierarchical algorithms and partitional algorithms. A partitional algorithm divides a data set into a single partition, whereas a hierarchical algorithm divides a data set into a sequence of nested partitions. In hierarchical clustering (Duda et al., 2000; Johnson, 1967), the goal is to find a hierarchy (generally represented by a tree) of partitions of data which may reveal interesting structure in the data at multiple levels of granularity. There are two basic approaches for generating a hierarchical clustering (Figure 4):

Agglomerative (bottom-up): Treat each point as a singleton cluster at the outset and then successively merge (or agglomerate) pairs of clusters until all clusters have been merged into a single cluster that contains all points (Sneath et al., 1973; KING, 1967; Gower, 1967). This requires defining a notion of cluster proximity.

Divisive (top-down): Start with one, all-inclusive cluster and then split clusters recursively until only singleton clusters of individual points remain. In this case, we need to decide how to split clusters (Dasgupta et al., 2006).

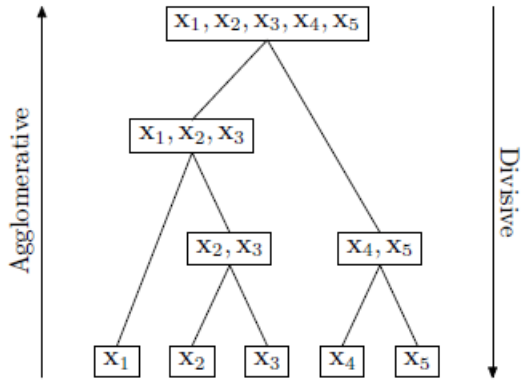


Figure 4: Agglomerative and Divisive hierarchical clustering

Agglomerative hierarchical clustering techniques are by far the most common, and the main focus of this work.

In this chapter, we discuss in detail hierarchical clustering. We discuss some of the major strengths of hierarchical clustering that makes the algorithms so popular in the machine learning community. We also discuss some of the major weaknesses of hierarchical algorithms, *i.e.* not being robust to noise or outliers. We also introduce some representation methods for hierarchical clustering. The ease of representation and the visual appeal of these representations is an important reason why hierarchical algorithms are so popular.

We then discuss in detail some of the most popular agglomerative clustering algorithms like *standard linkage algorithms, Wishart's Method, CURE, etc.* and describe their motivations, strengths and weaknesses. We also discuss briefly about divisive algorithms and suggested some of the reasons why these algorithms are not as popular as the agglomerative algorithms. However, we describe one, primarily divisive, algorithm *EigenCluster* owing to it's popularity in the machine learning community due to it's use of a highly innovative spectral algorithm while splitting clusters. We do an extensive experimental analysis of the robustness of all these methods by comparing their performance on various data sets.

2.1 Strengths and Weaknesses

Agglomerative algorithms are typically used when the underlying application, e.g., creation of a taxonomy, requires a hierarchy. In these fields, these algorithms are extremely popular as they are fast and their output is easy to interpret. Moreover, any valid measure of similarity can be used, in fact, the observations themselves are not required if we have a valid matrix of pairwise similarities. The number of clusters need not be specified beforehand and problems due to incorrect initialization and local minima do not arise (Frigui & Krishnapuram, 1997). Also, there have been some studies that suggest that these algorithms can produce better-quality clusters.

However, one of the main limitations of these methods is that they are not robust to

noise or outliers (Narasimhan et al., 2005; Balcan & Gupta, 2010a). This is due to several limitations:

***Lack of a Global Objective Function*:** Agglomerative hierarchical clustering cannot be viewed as globally optimizing an objective function. Instead, agglomerative hierarchical clustering techniques use various criteria to decide *locally*, at each step, which clusters should be merged (or split for divisive approaches) next. Though this approach yields clustering algorithms that avoid the difficulty of attempting to solve a hard combinatorial optimization problem and alleviate the problems with local minima or difficulties in choosing initial points, they cannot incorporate any *a priori* knowledge about the global shape or size of clusters which results in sensitivity to noise.

***Merging Decisions Are Final*:** Agglomerative hierarchical clustering algorithms tend to make good local decisions about combining two clusters since they can use information about the pairwise similarity of all points. However, once a decision is made to merge two clusters, it cannot be undone at a later time. This approach prevents a local optimization criterion from becoming a global optimization criterion. Thus, points incorrectly committed to a given (possibly unstable) cluster in the early stages cannot be moved to a different cluster afterwards.

Note: There are some techniques that attempt to overcome the limitation that merges are final. One approach attempts to fix up the hierarchical clustering by moving branches of the tree around so as to improve a global objective function.

In all agglomerative algorithms, a measure of similarity between pairs of points is assumed. Different schemes are distinguished by how they convert this similarity information into a measure of similarity between two clusters containing these points. For example, in *single linkage* (Sneath et al., 1973) the similarity between two clusters is the maximum similarity between points in these two different clusters whereas in *complete linkage* (KING, 1967), the similarity between two clusters is the minimum similarity between points in

these two different clusters. Using different measures for similarity between clusters may lead to different results. Thus, *a priori* knowledge of the right similarity measure is essential.

We further discuss specific strengths and weaknesses of different algorithms below.

2.2 Representation

A hierarchical clustering can be represented by either a picture or a list of abstract symbols. A picture of a hierarchical clustering is much easier for humans to interpret. A list of abstract symbols of a hierarchical clustering may be used internally to improve the performance of the algorithm. In this section, we discuss some common representations of hierarchical clusterings.

2.2.1 *n*-Tree

A hierarchical clustering is generally represented by a tree diagram. An n-tree is a simple hierarchically nested tree diagram that can be used to represent a hierarchical clustering. Let $S = \{x_1, x_2, ..., x_n\}$ be a set of objects. Then an n-tree on S is defined to be a set T of subsets of D satisfying the following conditions (Bobisud & Bobisud, 1972):

1. $S \in T$.
2. Empty set $\phi \in T$.
3. $\{x_i\} \in T, \forall i \in [n]$
4. if $A, B \in T$, then $A \cap B \in \{\phi, A, B\}$

A 5-tree is illustrated in Figure 5. The terminal nodes or leaves depicted by an open circle represent a single data point. The internal nodes depicted by a filled circle represent a group or cluster. n-trees are also referred to as nonranked trees (Murtagh, 1984). Tree diagrams, such as n-trees and dendrograms (discussed below), contain many indeterminacies. For example, the order of the internal nodes and the order of leaves can be interchanged.

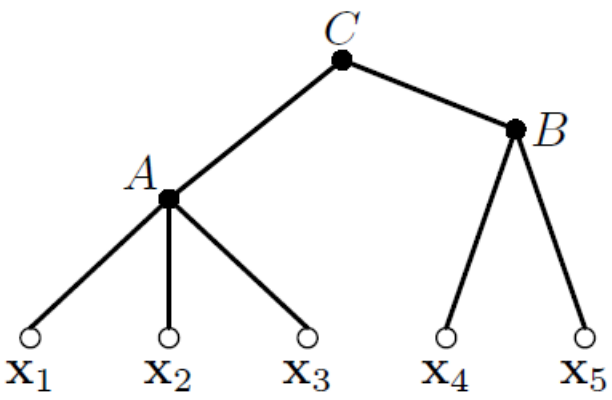


Figure 5: A 5-tree (n-tree of 5 points)

2.2.2 Dendrogram

A dendrogram is also called a valued tree (Arabie et al., 1996). A dendrogram is an n-tree in which each internal node is associated with a height satisfying the condition

$$h(A) \leq h(B) \Leftrightarrow A \subseteq B$$

for all subsets of points A and B if $A \cap B \neq \phi$, where $h(A)$ and $h(B)$ denote the heights of A and B respectively.

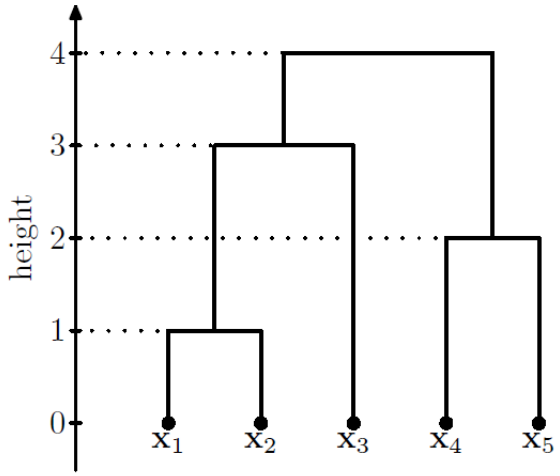


Figure 6: A dendrogram

As an illustration, Figure 6 shows a dendrogram with five data points. The dotted lines indicate the heights of the internal nodes. For each pair of data points (x_i, x_j), let h_{ij} be the

height of the internal node specifying the smallest cluster to which both x_i and x_j belong. Then a small value of h_{ij} indicates a high similarity between x_i and x_j. In the dendrogram given in Figure 6, for example, we have $h_{12} = 1, h_{23} = h_{13} = 3, and h_{14} = 4$.

The following ultrametric condition for height is a necessary and sufficient condition for a dendrogram.

$$h_{ij} \leq \max\{h_{ik}, hjk\}, \forall i, j, k \in [n]$$

Other representations of hierarchical structures include *Banner* (Rousseeuw, 1986), *pointer representation* (Sibson, 1973), *packed representation* (Sibson, 1973), icicle plot (Kruskal & Landwehr, 1983), loop plot (Kruskal & Landwehr, 1983) (Figure 7), etc. However, most of these representations are not generic suitable only for representing small data sets.

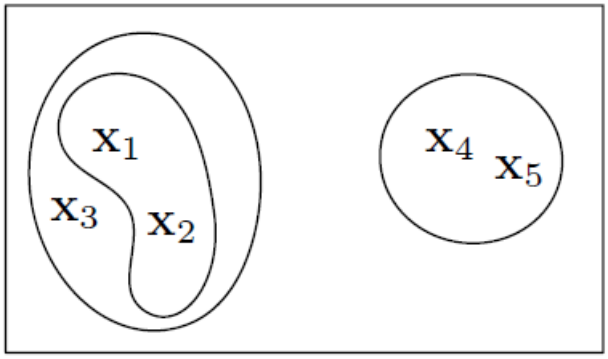


Figure 7: Loop Plot

According to different distance measures between clusters, agglomerative hierarchical methods can be subdivided into single-link methods, complete link methods, etc. The single, complete, average, and weighted average linkage methods are also referred to as graph methods, while Ward's method, the centroid method, and the median method are referred to as geometric methods, since in graph methods a cluster can be represented by a subgraph or interconnected points and in geometric methods a cluster can be represented by a center point.

2.3 Standard Agglomerative Linkage Algorithms

Agglomerative algorithms treat each point as a singleton cluster at the outset and then successively merge closest pairs of clusters until all clusters have been merged into a single cluster that contains all points.

Algorithm 1 describes the steps followed in any of the standard linkage based method.

Algorithm 1 Agglomerative Hierarchical Clustering

Input: similarity function $\mathcal{K}$, set of points S.

1. Start by assigning each item to it's own cluster. Let the similarities between the clusters equal the similarities between the items they contain.
2. Find the closest (most similar) pair of clusters $P, Q \subseteq S$ and merge them into a single cluster R.
3. Compute similarities between the new cluster R and each of the old clusters $C \subseteq S$.
4. Repeat steps 2 and 3 until all items are clustered into a single cluster of size n.

Output: Tree T on subsets of S.

Defining Similarity between Clusters

The key operation of the above algorithm is the computation of the similarity between two clusters, and it is the definition of cluster similarity that differentiates the various agglomerative hierarchical techniques that we discuss below.

2.3.1 Single Linkage

The single-link method (Florek et al., 1951; McQuitty, 1967) is one of the simplest hierarchical clustering methods. It is also known by other names, such as the nearest neighbor method, the minimum method, and the connectedness method (Johnson, 1967). The similarity of two clusters is defined as the *maximum similarity* between any two points in the two different clusters (Figure 8).

$$\mathcal{K}(R, C) = \max_{x \in R, x' in C} \mathcal{K}(x, x')$$

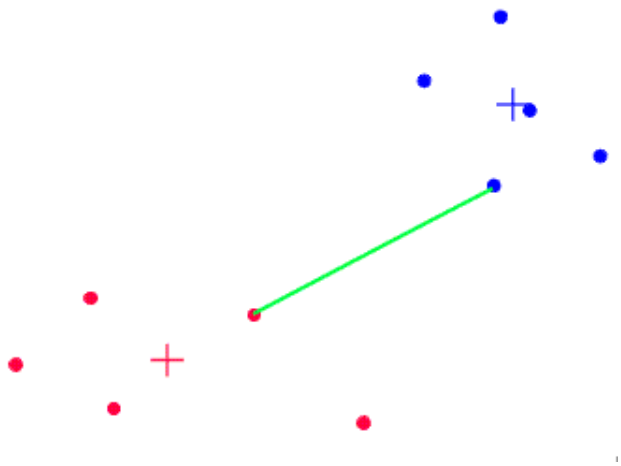


Figure 8: Single Linkage: The similarity of two clusters is defined as the *maximum similarity* between any two points in the two different clusters.

The single linkage method is invariant under monotone transformations (such as the logarithmic transformation) of the original data. It is good at handling non-elliptical shapes, but is sensitive to noise and outliers.

2.3.2 Complete Linkage

In complete linkage, the similarity of two clusters is defined as the *minimum similarity* between any two points in the two different clusters (Figure 9).

$$\mathcal{K}(R, C) = \min_{x \in R, x' in C} \mathcal{K}(x, x')$$

Complete linkage is also invariant under monotone transformations. It is less susceptible to noise and outliers, but it can break larger clusters and it favors spherical shapes.

2.3.3 Group Average Linkage

Average linkage is an intermediate approach between the single and complete link approaches. It is also referred as *UPGMA*, which stands for Unweighted Pair Group Method using arithmetic Averages (Jain & Dubes, 1981). The similarity of two clusters is defined as the *average pairwise similarity* among all pairs of points in the two different clusters

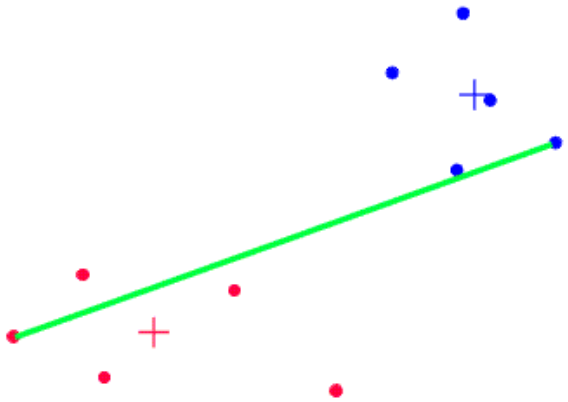


Figure 9: Complete Linkage: The similarity of two clusters is defined as the *minimum similarity* between any two points in the two different clusters.

(Figure 10).

$$\mathcal{K}(R, C) = \frac{1}{n_R n_C} \sum_{x \in R} \sum_{x' \in C} \mathcal{K}(x, x')$$

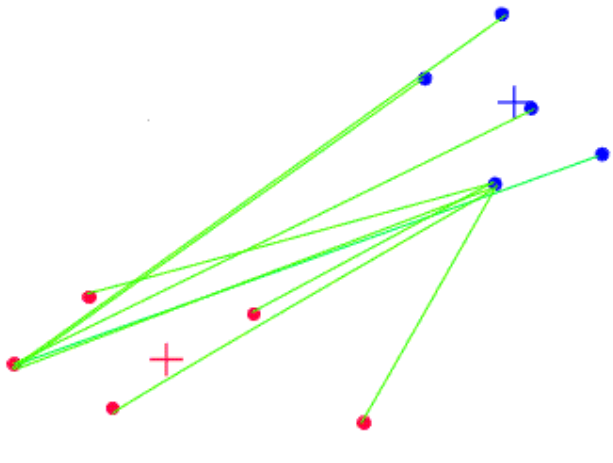


Figure 10: Average Linkage: The similarity of two clusters is defined as the *average pairwise similarity* among all pairs of points in the two different clusters.

In cases where cluster sizes vary, like complete linkage, average linkage also breaks larger clusters into parts. It does not work well in the presence of non spherical clusters or outliers.

2.3.4 Centroid Linkage

The centroid linkage method is also referred to as $UPGMC$, which stands for Unweighted Pair Group Method using Centroids (Jain & Dubes, 1981). It calculates the distance between two clusters by calculating the *distance between the centroids* of the two clusters (Figure 11). Let $\bar{x}_R = \frac{1}{n_R} \sum_{x \in R} x$ be the centroid of cluster R.

$$d(R, C) = \|\bar{x}_R - \bar{x}_C\|_2$$

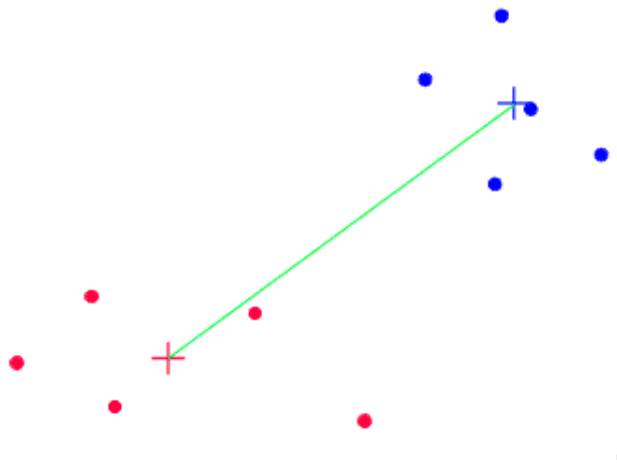


Figure 11: Centroid Linkage: The distance between two clusters is the *distance between the centroids* of the two clusters. The symbol $\times$ marks the centroids of the two clusters.

Centroid linkage is defined only for data in metric space and assumes that input data gives the distance from origin. It has a characteristic often considered bad that is not possessed by the other hierarchical clustering techniques that we have discussed: the possibility of **inversions**. Specifically, two clusters that are merged may be more similar (less distant) than the pair of clusters that were merged in a previous step. For the other methods, the distance between merged clusters monotonically increases (or is, at worst, non-increasing) as we proceed from singleton clusters to one all-inclusive cluster.

Another disadvantage is that if the sizes of the two groups to be merged are quite different, then the centroid of the new group will be very close to that of the larger group and may remain within that group (Everitt, 1993).

2.3.5 Median Linkage

The median linkage method is also referred to as $WPGMC$, which stands for Weighted Pair Group Method using Centroids (Jain & Dubes, 1981). It was first proposed by (Gower, 1967) in order to alleviate some disadvantages of the centroid method. In the centroid method, if the sizes of the two groups to be merged are quite different, then the centroid of the new group will be very close to that of the larger group and may remain within that group (Everitt, 1993). In the median method, the centroid of a new group is independent of the size of the groups that form the new group.

It calculates the distance between two clusters by calculating the weighted distance between the centroids of the two clusters.

$$d(R, C) = \|\tilde{x}_R - \tilde{x}_C\|_2$$

where $\tilde{x}_R = \frac{1}{2}(\tilde{x}_P + \tilde{x}_Q)$ if cluster R was created by combining clusters P and Q.

Median linkage is defined only for data in metric space and assumes that input data gives the distance from origin and also suffers from the possibility of **inversions**. Another disadvantage of the median method is that it is not suitable for measures such as correlation coefficients, since interpretation in a geometrical sense is no longer possible. Moreover, it breaks non-spherical clusters and is susceptible to noise.

2.3.6 Ward's Linkage

(Ward, 1963) proposed a hierarchical clustering procedure seeking to form the partitions in a manner that minimizes the loss of information associated with each merging. Usually, the information loss is quantified in terms of an error sum of squares (ESS) criterion, so Ward's method is often referred to as the minimum variance method. Thus, this method uses the same objective function as $K - means$ clustering.

It uses the incremental sum of squares, *i.e.* the increase in the total within-cluster sum of squares as a result of joining two clusters, which is defined as the sum of the

squares of the distances between all objects in the cluster and the centroid of the cluster. Let $\bar{x}_R = \frac{1}{n_R} \sum_{x \in R} x$ be the centroid of cluster R.

$$d(R, C) = \sqrt{\frac{2n_R n_C}{n_R + n_C}} \|\bar{x}_R - \bar{x}_C\|_2$$

While it may seem that this feature makes Ward's method somewhat distinct from other hierarchical techniques, it can be shown mathematically that Ward's method is very similar to the group average method when the proximity between two points is taken to be the square of the distance between them.

2.4 (Generalized) Wishart's Method

(Wishart, 1969) proposed a new robust algorithm to overcome the insensitivity of single linkage to data density which causes *chaining* (low density noisy points tying larger cohesive clusters together). At each step, Wishart's method avoids chaining by discarding as noise regions with density lower than a particular threshold before merging clusters, ensuring regions of high density become available for linkage earlier than regions of low density. When connecting points at distance r from each other, only those points are considered that have at least $k(> 2)$ neighbors within distance r.

However, it was unclear how to choose the input parameter for the algorithm till (Chaudhuri & Dasgupta, 2010) proposed a generalization of Wishart's method and gave a theoretical basis for choosing the input parameter. This generalized algorithm is described in Algorithm 2.

Note that for this algorithm $\alpha \geq 1$ and is equivalent to Wishart's Method when $\alpha = 1$.

Unfortunately, both these methods are not robust to *sparse noise* (a large fraction of the points in the data have very high similarity to a few points from other target clusters even though most of the nearest neighbors are from their own target cluster).

Algorithm 2 Generalized Wishart's Method

Input: similarity function $\mathcal{K}$, set of points S, $k \geq 2$, $\alpha \geq 1$.

1. For each $x \in S$, set $r_k(x) = \inf\{r : B(r,k)$ contains k data points $\}$.
2. As r grows from 0 to ∞.
   (a) Construct a graph G_r with nodes $\{x : r_k(x) \leq r\}$. Include edge (x,y) if $\|x-y\|_{\leq} \alpha r$.
   (b) Let C_r be the connected component of G_r.

Output: Tree T on subsets of S.

2.5 BIRCH

(Zhang et al., 1996) proposed an agglomerative hierarchical algorithm, called *BIRCH* (Balanced Iterative Reducing and Clustering using Hierarchies), for clustering very large numerical data sets in Euclidean spaces. It is also the first clustering algorithm in the database area that takes account of noise. In the algorithm of *BIRCH*, a clustering feature (CF) vector is used to summarize the information of each cluster. Given a cluster C of a d-dimensional data set, the CF vector for C is a triple defined as

$$CF(C) = (|C|, S_1, S_2)$$

where S_1 and S_2 are d-dimensional vectors defined as

$$\begin{aligned} S_1 &= \sum_{x \in C} x = \left(\sum_{x \in C} x_1, \sum_{x \in C} x_2, \cdot, \cdot, \cdot, \sum_{x \in C} x_d \right) \\ S_2 &= \sum_{x \in C} x^2 = \left(\sum_{x \in C} x_1^2, \sum_{x \in C} x_2^2, \cdot, \cdot, \cdot, \sum_{x \in C} x_d^2 \right) \end{aligned}$$

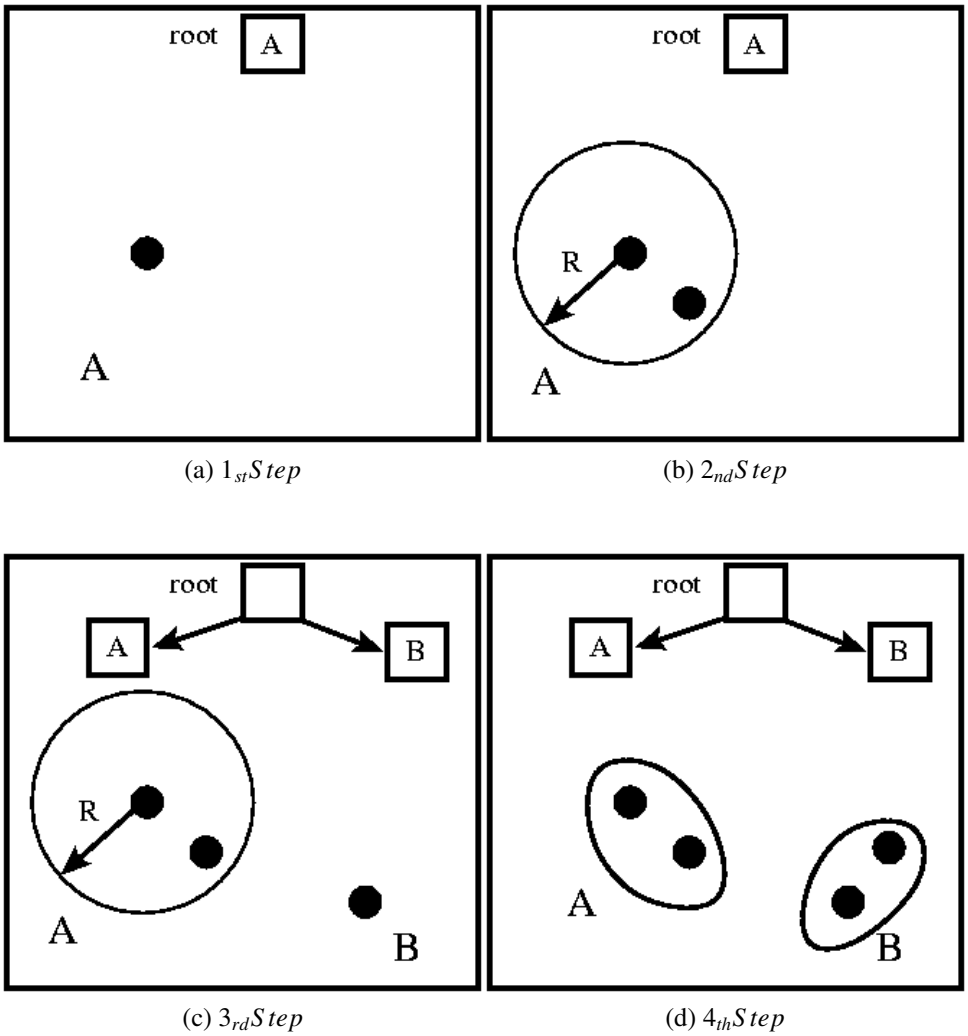
where $x_j (1 \leq j \leq d)$ is the value of the j_{th} attribute of x.

At the beginning, a CF tree is built dynamically as new data objects are inserted. A CF tree has three parameters:

1. B: branching factor.
2. L: leaf factor.

3. T: threshold.

Each nonleaf node contains at most B subnodes of the form $[CF_i,\ child_i]$, each leaf node contains at most L entries of the form $[CF_i]$, and the diameter of each entry in a leaf node has to be less than T (Figure 12). Outliers or noise are determined by considering the density of each entry in leaf nodes; *i.e.* low-density entries of leaf nodes are treated as outliers. Potential outliers are written out to disk in order to reduce the size of the tree. At certain points in the process, outliers are scanned to see if they can be reabsorbed into the current tree without causing the tree to grow in size.



(a) $1_{st} Step$ (b) $2_{nd} Step$

(c) $3_{rd} Step$ (d) $4_{th} Step$

Figure 12: *BIRCH*: The idea of CF Tree

After the CF tree is built, an agglomerative hierarchical clustering algorithm is applied directly to the nodes represented by their CF vectors. Then for each cluster, a centroid is obtained. Finally, a set of new clusters is formed by redistributing each data point to it's nearest centroid.

BIRCH works well when clusters are of convex or spherical shape and uniform size. However, it is unsuitable when clusters have different sizes or non-spherical shapes (Guha et al., 1998). Further, is only applicable for similarity measures that are metric and cannot be used in general.

2.6 CURE

(Guha et al., 1999) proposed an agglomerative hierarchical clustering algorithm called *CURE* (Clustering Using REpresentatives) that can identify non-spherical shapes in large databases and wide variance in size. A combination of random sampling and partitioning is used in CURE in order to handle large databases.

In *CURE*, a constant number c of *well scattered* points in a cluster are chosen and shrunk towards the centroid of the cluster by a fraction α which are then are used as *representatives* of the cluster which capture it's shape and extent (Figure 13). The clusters with the closest pair of representative points are the clusters that are merged at each step as described in Algorithms 3 and 4.

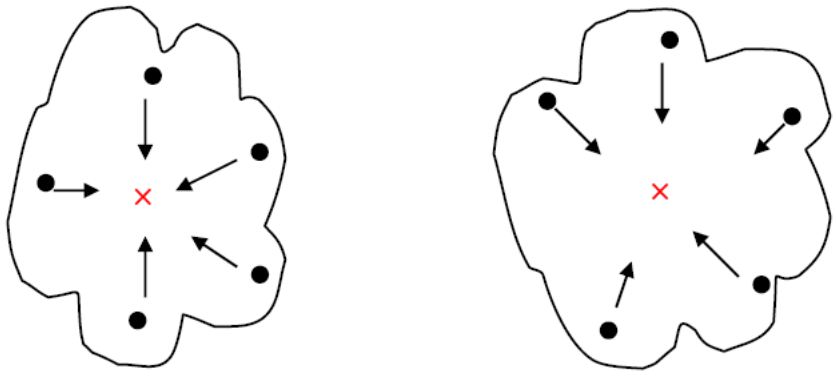


Figure 13: Shrinking *representatives* towards center of the cluster.

For each cluster u, we define:

***u.mean*:** mean of the points in the cluster u.

***u.rep*:** the set of c representative points of cluster u.

***u.closest*:** the cluster closest to u.

Algorithm 3 CURE

Input: set of points S, number of desired clusters $k > 1$.

1. Insert all points into a $k-d$ tree T.
2. Initially for each $u \in S$
   (a) $u.mean = u$
   (b) $u.rep = u$
   (c) Find $u.closest$ for each u and then insert each cluster into a min-heap Q.
3. While $size(Q) > k$
   (a) Remove the top (smallest) element of Q (say u) and merge it with it's closest cluster $u.closest$ (say v) using Algorithm 4 to get cluster w.
   (b) Delete $u.rep$ and $v.rep$ from T.
   (c) Delete u and v from Q.
   (d) Insert $w.rep$ to T.
   (e) For all clusters x in Q, update $x.closest$ and relocate x.
   (f) Update $w.closest$ and insert w to Q.

Output: Heap Q representing the k clusters and hierarchy H as the subset of S.

Note that when α is very low, the scattered points are shrunk very little and thus $CURE$ degenerates to the single linkage algorithm. $CURE$ behaves similar to traditional centroid-based hierarchical algorithms for values of α between 0.8 and 1 since the representative points end up close to the center of the cluster. Thus, $[0.2 - 0.7$ is a good range of values for α to identify non-spherical clusters while dampening the effects of outliers.

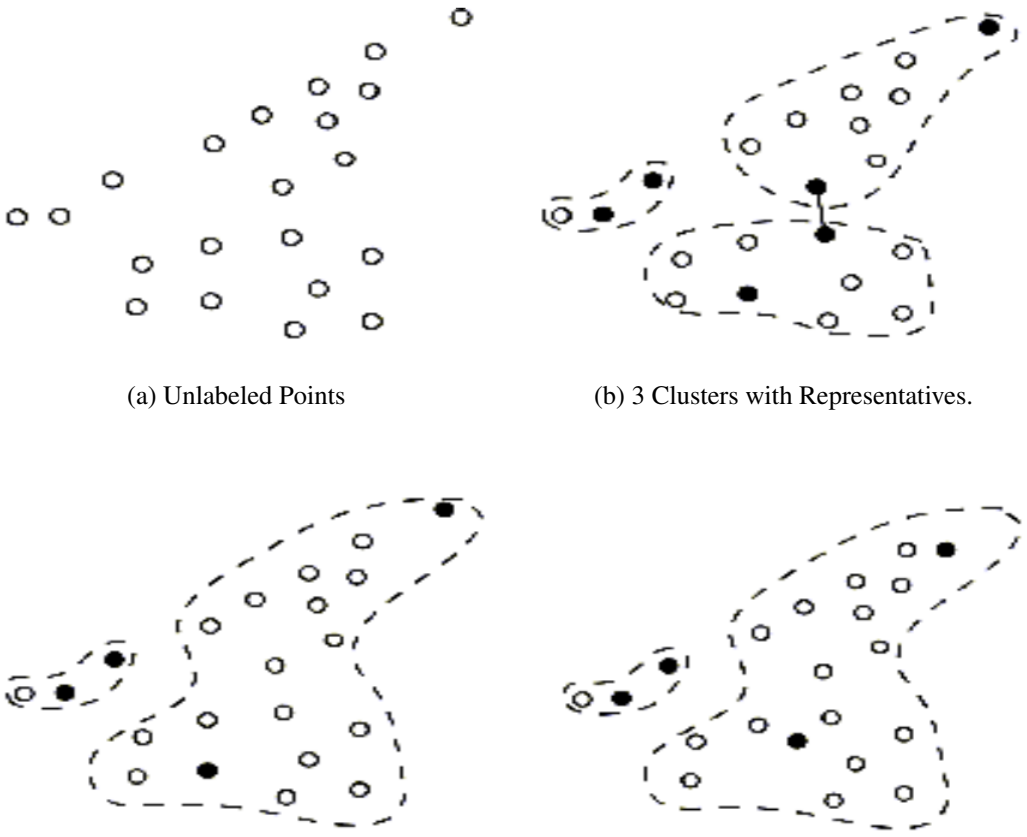
Like Wishart's methods, $CURE$ can also be shown to fail in case of sparse noise.

Algorithm 4 CURE Merge

Input: clusters u and v, number of representatives $c \geq 1$, shrink factor $\alpha = [0, 1]$.

1. Merge points in u and v to get a new cluster w.
2. Compute $w.mean$ from $u.mean$ and $v.mean$.
3. Find a set s of c scattered points as follows:
   (a) Choose the first scattered point as the point in w farthest from $w.mean$.
   (b) Chose subsequent scattered points as points in w farthest from the previously chosen scattered points in s.
4. Compute $w.rep$ by shrinking points in s towards $w.mean$ by a fraction α.

Output: Cluster w.



(a) Unlabeled Points

(b) 3 Clusters with Representatives.

(c) Merged Clusters with closest Representatives.

(d) Shrink Representatives towards Mean.

Figure 14: *CURE* Example

2.7 Divisive Hierarchical Algorithms

The divisive hierarchical method proceeds the opposite way of the agglomerative hierarchical method. Initially, all the data points belong to a single cluster. The number of clusters is increased by one at each stage of the algorithm by dividing an existing cluster into two clusters according to some criteria. A divisive hierarchical method may be adopted in which a single cluster is subdivided into smaller and smaller clusters. Divisive hierarchical clustering methods are essentially of two types: **monothetic** and **polythetic** (Everitt, 1993; Willett, 1988). A monothetic method divides the data on the basis of the possession or otherwise of a single specified attribute, while a polythetic method divides the data based on the values taken by all attributes.

It is not feasible to enumerate all possible divisions of a large (even moderate) cluster C to find the optimal partition, since there are $2^{|C|-1} - 1$ nontrivial different ways of dividing the cluster C into two clusters (Edwards & Cavalli-Sforza, 1965). (Scott & Symons, 1971) have proposed an improved algorithm that requires examination of $2^d - 2$ partitions by assigning points in the hyperplane to the two clusters being considered, where d is the dimensionality of the data. Except for low-dimensional data, this algorithm is also very time-consuming. In fact, it turns out that the problem of finding an optimal bipartition for some clustering criteria is NP-hard (Arabie et al., 1996).

Another problem with divisive hierarchical algorithms is monotonicity, to be specified below. In a divisive hierarchy, one cluster is divided at a time, so what is the next cluster to be divided? This depends on the definition of a level. Such a level must be meaningful and monotone, which means that no subcluster may have a larger level than the level of it's parent cluster.

However, it is possible to construct divisive hierarchical algorithms that do not need to consider all divisions and are monothetic. An algorithm called *DIANA*(*DIvisiveANAlysis*) (Kaufman & Rousseeuw, 1990) is a divisive hierarchical clustering algorithm. It was based on the idea of (Macnaughton-Smith et al., 1964).

2.8 *EigenCluster*

EigenCluster (Cheng et al., 2006) combines top-down and bottom-up techniques to create both a hierarchy and a flat clustering (Figure 15). In the top-down phase, it divides clusters using a spectral algorithm (Kannan et al., 2004) to form a tree whose leaves are the objects. The input to this phase is a set of objects whose pairwise similarities or distances are given, or can be easily computed from the objects themselves. The algorithm recursively partitions a cluster into two smaller sets until it arrives at singletons. This is followed by the bottom-up phase starting with each leaf of the tree in it's own cluster and merging clusters going up the tree using uses a dynamic program to find the optimal tree-respecting clustering for a given a natural objective function, *e.g.* min-diameter, min-sum, etc. The final clusters form a partition of the dataset and are tree-respecting clusters, *i.e.* they are subtrees rooted at some node of the tree.

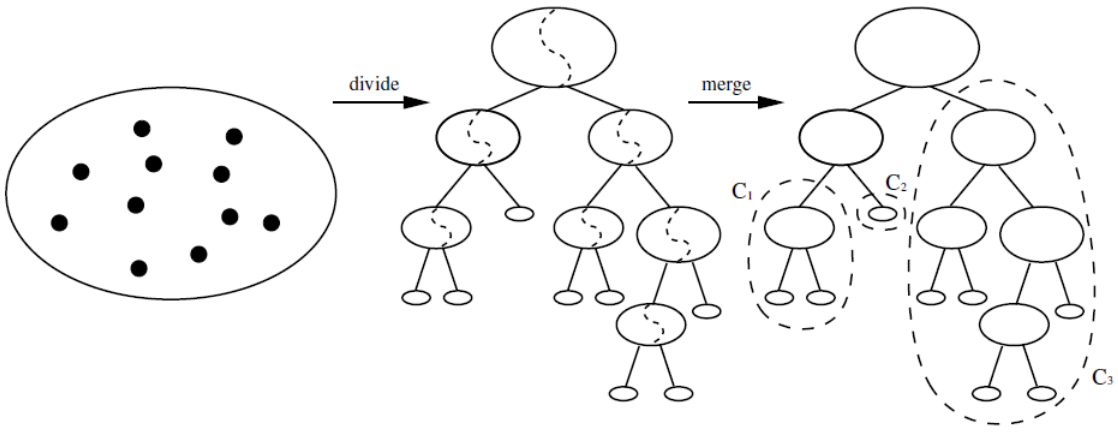


Figure 15: The Divide-and-Merge methodology

Note that the cut computed using the second eigenvector of M is not the cut of minimum conductance; finding such a cut is NP-hard. However, the conductance of the computed cut is not much worse than the minimum conductance (OPT) cut and is within $\sqrt{2OPT}$ (Sinclair & Jerrum, 1988).

In the merge phase, dynamic programming is used to compute the optimal clustering

Algorithm 5 Divide Phase

Input: set of points S represented by $n \times m$ matrix A.

1. Compute similarity matrix M based on dot product, *i.e.* $M = AA^T$.
2. Normalize M so that all row sums are 1.
3. Compute v, an approximation of the second eigenvector of M.
4. Compute a set of $n - 1$ cuts using the ordering of the points in v.
5. Compute the *best cut* as the cut of minimum conductance and split S into C and $S \setminus C$.
6. Recurse on C and $S \setminus C$.

Output: Tree T on subsets of S

in the tree for many standard objective functions. Assuming the objective function to maximize is g, the dynamic program $OPT - TREE$ computes a clustering $C_{OPT-TREE}$ in the tree T such that $g(C_{OPT-TREE})$ is greater than any other clustering in the tree. Let u be a node in tree T and l_u, r_u be the left and the right child of u respectively. Let i be number of desired sub-clusters of u. Then, the dynamic program $OPT - TREE(u, i)$ has the following recursion:

$$\begin{aligned} OPT - TREE(u, 1) &= u \\ OPT - TREE(u, i) &= OPT - TREE(l_u, j) \bigcup OPT - TREE(r_u, i - j) \end{aligned}$$

where

$$j = \arg\min_{1<j<i} \quad g\left(OPT - TREE(l_u, j) \bigcup OPT - TREE(r_u, i - j)\right)$$

The optimal clustering for leaf nodes can be computed first and then the optimal clustering can be efficiently determined for any interior node. Let R_T be the root of the tree T. Then, $OPT - TREE(R_t, k)$ gives the optimal k-clustering.

2.9 Discussion

In this chapter, we discussed in detail hierarchical clustering. We discussed some of the major strengths of hierarchical clustering that makes the algorithms so popular in the machine learning community. We also discussed some of the major weaknesses of hierarchical algorithms, *i.e.* not being robust to noise or outliers. Since real world data is almost always noisy, this weakness may sometimes prohibit the use of hierarchical algorithms. We further discuss the robustness of some of the most popular hierarchical algorithms in Chapter 3. In Chapter 7, we see experimentally that most standard hierarchical algorithms are in fact not useful and fail miserably while clustering real-world noisy data sets.

We also introduced some representation methods for hierarchical clustering. The ease of representation and the visual appeal of these representations is an important reason why hierarchical algorithms are so popular.

We also discussed in detail some of the most popular agglomerative clustering algorithms like *standard linkage algorithms, Wishart's Method, CURE, etc.* and described their motivations, strengths and weaknesses. We also discussed briefly about divisive algorithms and suggested some of the reasons why these algorithms are not as popular as the agglomerative algorithms. However, we described one, primarily divisive, algorithm *EigenCluster* owing to it's popularity in the machine learning community due to it's use of a highly innovative spectral algorithm while splitting clusters.

We note here that most of the algorithms discussed above lack any theoretical guarantees for their correctness and robustness since theoretical analysis, if done at all, has typically involved either making strong assumptions about the uniformity of clusters or else optimizing distance-based objective functions only secondarily related to the true goals.

CHAPTER III

ROBUSTNESS OF HIERARCHICAL ALGORITHMS

In this chapter we analyze the performance of different hierarchical algorithms discussed in Chapter 2. We look at both theoretical and experimental results in order to understand how different algorithms behave under different conditions such as structure of data, presence of noise, incompleteness in data, etc.

In order to formally analyze correctness of algorithms we use the framework introduced by (Balcan et al., 2008). In this framework, it is assumed there is some target clustering (much like a k-class target function in the multi-class learning setting) and an algorithm is said to correctly cluster data satisfying some property P if on any data set having that property P, the algorithm produces a tree such that the target is some pruning of the tree. We use this framework to define a model to analyze the correctness of different algorithms. We describe this in detail in Section 3.1 and describe some of the relevant properties in Section 3.2.

In Section 3.3.1, we propose two new natural properties of data in which all points in the data set are allowed to be affected by some amount of noise called the good neighborhood properties. These properties capture extremely well various forms of real world noise like presence of outliers, incompleteness of data, data corruption, transmission errors, etc. The good neighborhood property roughly says that after a small number of extremely malicious points have been removed, for the remaining points in the data set, most of their nearest neighbors are from their target cluster.

3.1 Framework

(Balcan et al., 2008) proposed a new general framework for analyzing clustering from similarity information that addresses the question of what properties of a similarity measure

are sufficient to cluster accurately and by what kinds of algorithms. The framework can be viewed as an analog for clustering of discriminative models for supervised classification (*i.e.* the Statistical Learning Theory framework and the *PAC* learning model), where the goal is to cluster accurately given a *property* or relation the similarity function is believed to satisfy with respect to the ground truth clustering. More specifically the framework is analogous to that of data-dependent concept classes in supervised learning, where conditions such as the large margin property have been central in the analysis of kernel methods.

In their work, they developed a theoretical approach to analyzing clustering that is able to talk about *accuracy* of a solution produced without resorting to a generative model for the data. In particular, they ask the question *what natural properties of a pairwise similarity function are sufficient to allow one to* ***cluster*** *well?* which relates closely to work on similarity functions in the context of Supervised Learning that asks *what natural properties of a given kernel (or similarity) function K are sufficient to allow one to* ***learn*** *well?* (Herbrich, 2002; Cristianini et al., 2001; Scholkopf et al., 2004; Balcan & Blum, 2006). To study this question they develop a theoretical framework which can be thought of as a discriminative (*PAC* style) model for clustering, though the basic object of study, rather than a concept class, is a *property* of the similarity function $\mathcal{K}$ in terms of it's relation to the target.

The setup we use in our work to analyze different algorithms follows directly from this framework.

3.1.1 Formal Setup

We consider a clustering problem (S, ℓ) specified as follows. Assume we have a data set S of n objects. Each $x \in S$ has some (unknown) "ground-truth" label $\ell(x)$ in $Y = \{1, \ldots, k\}$, where we will think of k as much smaller than n. We let $C_i = \{x \in S : \ell(x) = i\}$ denote the set of points of label i (which could be empty), and denote the target clustering as $C = \{C_1, \ldots, C_k\}$. For example, in a data set of bitmaps of handwritten digit, the digit on

each bitmap represents it's *ground-truth* label and the cluster formed by all the bitmaps of the same digit is the target cluster for that digit.

Given another proposed clustering h, $h : S \to Y$, we define the error of h with respect to the target clustering using *Classification Error* (Meila & Heckerman, 2001) which is defined to be

$$err(h) = \min_{\sigma \in \mathcal{S}_k} \left[\Pr_{x \in S} \left[\sigma(h(x)) \neq \ell(x) \right] \right], \tag{16}$$

where $\mathcal{S}_k$ is the set of all permutations on $\{1, \ldots, k\}$.

Equivalently, the error of a clustering $C' = \{C'_1, \ldots, C'_k\}$ is

$$\min_{\sigma \in \mathcal{S}_k} \frac{1}{n} \sum_i |C_i - C'_{\sigma(i)}| \tag{17}$$

First, for each cluster in C, a *best match* is found in C' and then the error is computed as the fraction of points misclassified in matched clusters. This is a natural way of comparing clusterings as it gives the probability a point chosen at random from S is labeled incorrectly in C'. [1]

Hierarchical algorithms generate a hierarchy rather than a single clustering as output. (Balcan et al., 2008) have shown that a hierarchical output is necessary to analyze non-trivial properties of the similarity function since there may exist multiple prunings of interest in the hierarchy. An algorithm is considered successful if the target clustering is *a pruning* of the computed hierarchy. For our experimental analysis, we compare all the prunings of the hierarchy to the target clustering of the data set and report the clustering of lowest *Classification Error*.

The goal of a hierarchical algorithm is to produce a hierarchical clustering that contains a pruning that is close to the target clustering. Formally, the goal of the algorithm is to produce a hierarchical clustering: that is, a tree on subsets such that the root is the set S, and the children of any node S' in the tree form a partition of S'. The requirement is that there must exist a *pruning* h of the tree (not necessarily using nodes all at the same

[1] A detailed discussion of Classification Error is done in Chapter B

level) that has error at most ϵ. In many applications (e.g. document clustering) this is a significantly more user-friendly output than the list model. Note that any given tree has at most 2^{2k} prunings of size k (Knuth, 1997).

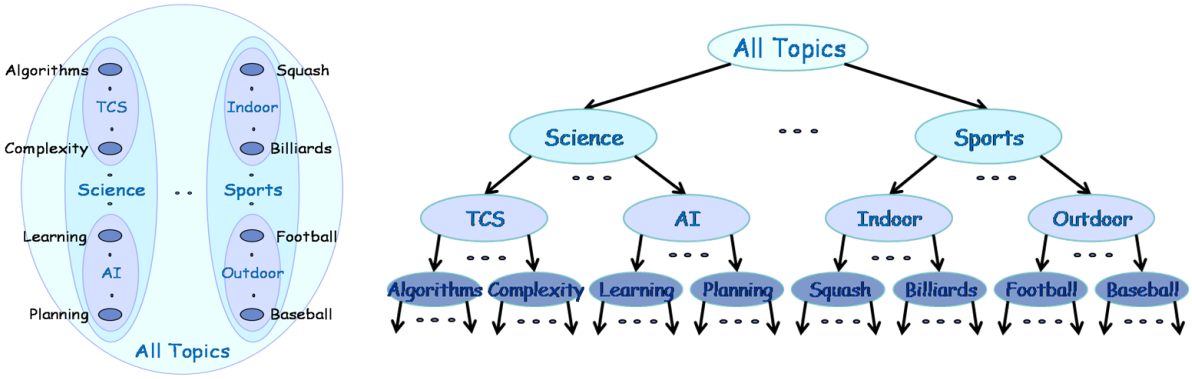


Figure 16: Consider a document clustering problem. Assume that data lies in multiple regions Algorithms, Complexity, Learning, Planning, Squash, Billiards, Football, Baseball. Suppose that $\mathcal{K}(x, y) = 0.999$ if x and y belong to the same inner region; $\mathcal{K}(x, y) = 3/4$ if $x \in$ Algorithms and $y \in$ Complexity, or if $x \in$ Learning and $y \in$ Planning, or if $x \in$ Squash and $y \in$ Billiards, or if $x \in$ Football and $y \in$ Baseball; $\mathcal{K}(x, y) = 1/2$ if x is in (Algorithms or Complexity) and y is in (Learning or Planning), or if x is in (Squash or Billiards) and y is in (Football or Baseball); define $\mathcal{K}(x, y) = 0$ otherwise. Both clusterings {Algorithms $\cup$ Complexity $\cup$ Learning $\cup$ Planning, Squash $\cup$ Billiards, Football $\cup$ Baseball} and {Algorithms$\cup$Complexity, Learning$\cup$Planning, Squash$\cup$Billiards$\cup$Football$\cup$Baseball} satisfy the strict separation property.

(Balcan et al., 2008) have shown that this type of output is necessary in order to be able to analyze non-trivial properties of the similarity function since there may exist multiple prunings of interest in the hierarchy. For example, even if the similarity function satisfies the requirement that all points are more similar to all points in their own cluster than to any point in any other cluster (this is called the strict separation property) and even if we are told the number of clusters, there can still be multiple different clusterings that satisfy the property. In particular, one can show examples of similarity functions and two significantly different clusterings of the data satisfying the strict separation property. See Figure 16 for an example. However, under the strict separation property, there is a single hierarchical decomposition such that any consistent clustering is a pruning of this tree. This motivates

clustering in the tree model and this is the model we consider in this work as well.

3.2 Properties of Similarity Function $\mathcal{K}$

We describe here some natural properties of the similarity functions that we discuss in this work. We start with the simple strict separation property and some other properties of interest introduced in (Balcan et al., 2008) and give some correctness results under these properties. We then define the *good neighborhood* property which is an interesting and natural generalization of it.

Note: *The proofs of Theorems 1, 2, 3 and 4 are presented in Appendix A.*

3.2.1 Strict Separation

The strict separation property implies that all clusters are well separated, *i.e.* all points belonging to the same cluster are more similar to each other than to any point outside their own cluster.

Property 1. *The similarity function $\mathcal{K}$ satisfies* **strict separation** *for the clustering problem (S, ℓ) if for all $x \in S$ with $x' \in C(x)$ and $x'' \notin C(x)$ we have $\mathcal{K}(x, x') > \mathcal{K}(x, x'')$.*

Given a similarity function satisfying the strict separation property (see Figure 16 for an example), we can efficiently construct a tree such that the ground-truth clustering is a pruning of this tree using the single linkage algorithm as formalized in Theorem 1 (Balcan et al., 2008).

Theorem 1. *Let $\mathcal{K}$ be a symmetric similarity function satisfying the strict separation property. Then we can efficiently construct a tree using single linkage algorithm such that the ground-truth clustering is a pruning of this tree.*

Moreover, almost any of the standard linkage based algorithms (*e.g.*, single linkage, average linkage, or complete linkage) would work well under this property.

Note: Strict separation property does not guarantee that all the cutoffs for different points x are the same, so single linkage would not necessarily have the right clustering if it just

stopped once it has k clusters; however the target clustering will provably be a pruning of the final single linkage tree; this is why we define success based on prunings.

The strict separation property is generalized by the ν-strict separation property which takes into account the presence of (νn) misbehaving points.

Property 2. *The similarity function $\mathcal{K}$ satisfies* **ν-strict separation** *property for the clustering problem (S, ℓ) if for some $S' \subseteq S$ of size $(1-\nu)n$, $\mathcal{K}$ satisfies strict separation property for (S', ℓ). That is, for all $x, x', x'' \in S'$ with $x' \in C(x)$ and $x'' \notin C(x)$ we have $\mathcal{K}(x, x') > \mathcal{K}(x, x'')$.*

So, in other words we require that the strict separation is satisfied after a (νn) number of bad points have been removed.

3.2.2 Max Stability

We begin with a stability property that relaxes strict separation and asks that the ground truth clustering be stable in a certain sense.

Property 3. *The similarity function $\mathcal{K}$ satisfies* **max stability** *property for the clustering problem (S, ℓ) if for all target clusters $C_r, C_{r'}, r \neq r', forall A \subset C_r, A' \subseteq C_{r'}$, we have*

$$\mathcal{K}_{max}(A, C_r \backslash A) > \mathcal{K}_{max}(A, A')$$

This implies that no subset A of some target cluster C_r would prefer to join another subset A' of some $C_{r'}$ if "prefer" is defined according to maximum similarity between pairs.

The max stability property characterizes the desiderata for single-linkage in that it is both necessary and sufficient for single-linkage to produce a tree such that the target clustering is a pruning of the tree as formalized in Theorem 2 (Balcan et al., 2008).

Theorem 2. *For a symmetric similarity function $\mathcal{K}$, max stability(Property 3) is a necessary and sufficient condition for single linkage to produce a tree such that the ground truth clustering is a pruning of this tree.*

3.3 Average Stability

A perhaps more natural notion of stability, and general as compared to max stability, is to define ”prefer” with respect to the average. The result is a notion much like stability in the ”stable marriage” sense, but for clusterings. In particular, the following properties:

Property 4. *The similarity function $\mathcal{K}$ satisfies* **strong stability** *property for the clustering problem (S, ℓ) if for all target clusters $C_r, C_{r'}, r \neq r', forall A \subset C_r, A' \subseteq C_{r'}$, we have*

$$\mathcal{K}(A, C_r \backslash A) > \mathcal{K}(A, A')$$

Given a similarity function satisfying the average stability properties, we can efficiently construct a tree such that the ground-truth clustering is a pruning of this tree using the average linkage algorithm as formalized in Theorem 3 and Theorem 4 (Balcan et al., 2008).

Theorem 3. *Let $\mathcal{K}$ symmetric similarity function $\mathcal{K}$ satisfying strong stability.Then average linkage algorithm constructs a binary tree such that the ground truth clustering is a pruning of this tree.*

Property 5. *The similarity function $\mathcal{K}$ satisfies* **weak stability** *property for the clustering problem (S, ℓ) if for all target clusters $C_r, C_{r'}, r \neq r', forall A \subset C_r, A' \subseteq C_{r'}$, we have*

- *If $A' \subset C_{r'}$, then either $\mathcal{K}(A, C_r \backslash A) > \mathcal{K}(A, A')$ or $\mathcal{K}(A', C_{r'} \backslash A') > \mathcal{K}(A', A)$*
- *If $A' = C_{r'}$, then $\mathcal{K}(A, C_r \backslash A) > \mathcal{K}(A, A')$*

We can interpret weak stability as saying that for any two clusters in the ground truth, there does not exist a subset A of one and subset A' of the other that are more attracted to each other than to the remainder of their true clusters (with technical conditions at the boundary cases) much as in the classic notion of stable marriage. Strong stability asks that both be more attracted to their true clusters.

Theorem 4. *Let $\mathcal{K}$ symmetric similarity function $\mathcal{K}$ satisfying weak stability.Then average linkage algorithm constructs a binary tree such that the ground truth clustering is a pruning of this tree.*

Note that if the similarity function $\mathcal{K}$ is asymmetric then even if strong stability is satisfied the average linkage algorithm may fail.

3.3.1 Good Neighborhood

We present here two new properties of the similarity function $\mathcal{K}$ which not only allow some points to be malicious, but also allow for each point to have some bad immediate neighbors as long as most of it's immediate neighbors are good. Thus all points in the data set are allowed to be affected by some amount of noise in the good neighborhood properties.

Property 6. *The similarity function $\mathcal{K}$ satisfies α-**good neighborhood** property for the clustering problem (S, ℓ) if for all points x we have that all but αn out of their $n_{C(x)}$ nearest neighbors belong to the cluster $C(x)$.*

This implies that for all points in the data set, most of their nearest neighbors are from their target cluster.

Note: α-good neighborhood property is different from the ν-strict separation property. For the ν-strict separation property we can have up to νn bad points that can misbehave; in particular, these νn bad points can have similarity 1 to *all* the points in S; however, once we remove these points the remaining points are more similar to points in their own cluster than to points in other cluster. On the other hand, for the α-good neighborhood property we require that for all points x all but αn out of their $n_{C(x)}$ nearest neighbors belong to the cluster $C(x)$. (So we cannot have a point that has similarity 1 to all the points in S.) Note however that different points might misbehave on different αn neighbors.

We define below the property that generalizes both the ν-strict separation property and the α-good neighborhood.

Property 7. *The similarity function $\mathcal{K}$ satisfies (α, ν)-**good neighborhood** property for the clustering problem (S, ℓ) if for some $S' \subseteq S$ of size $(1-\nu)n$, $\mathcal{K}$ satisfies α-good neighborhood for (S', ℓ). That is, for all for all points $x \in S'$ we have that all but αn out of their $n_{C(x)}$ nearest neighbors belong to the cluster $C(x)$.*

This implies that after a small number of extremely malicious points have been removed, for the remaining points in the data set, most of their nearest neighbors are from their target cluster.

It is easy to see that:

Fact 1. *If the similarity function $\mathcal{K}$ satisfies the α-good neighborhood property for the clustering problem (S, ℓ), then $\mathcal{K}$ also satisfies the $(\alpha, 0)$-good neighborhood property for the clustering problem (S, ℓ).*

Fact 2. *If the similarity function $\mathcal{K}$ satisfies the ν-strict separation property for the clustering problem (S, ℓ), then $\mathcal{K}$ also satisfies the $(0, \nu)$-good neighborhood property for the clustering problem (S, ℓ).*

These two properties capture extremely well various forms of real world noise like presence of outliers, incompleteness of data, data corruption, transmission errors, etc.

3.4 Robustness of Standard Linkage Algorithms

In Chapter 2, we suggested that most of the standard linkage algorithms are not robust to noise. Here we give some justification to that claim. We show that the standard linkage algorithms fail if the data has noise even though the data may satisfy other natural properties, *e.g.* good neighborhood property.

We can show that if the similarity function slightly deviates from the strict separation condition, then all the standard agglomerative algorithms will fail even though the data may satisfy other natural properties, *e.g.* good neighborhood property. We show an example where if the data satisfies the good neighborhood property, then essentially most of the standard linkage based algorithms will fail.

We slightly modify the example in Figure 16, by adding a little bit of noise, to form links of high similarity between points in different inner blobs, we can show that many

of the classic linkage based algorithms will perform poorly[2]. See Figure 17 for a precise description of the similarity measure.

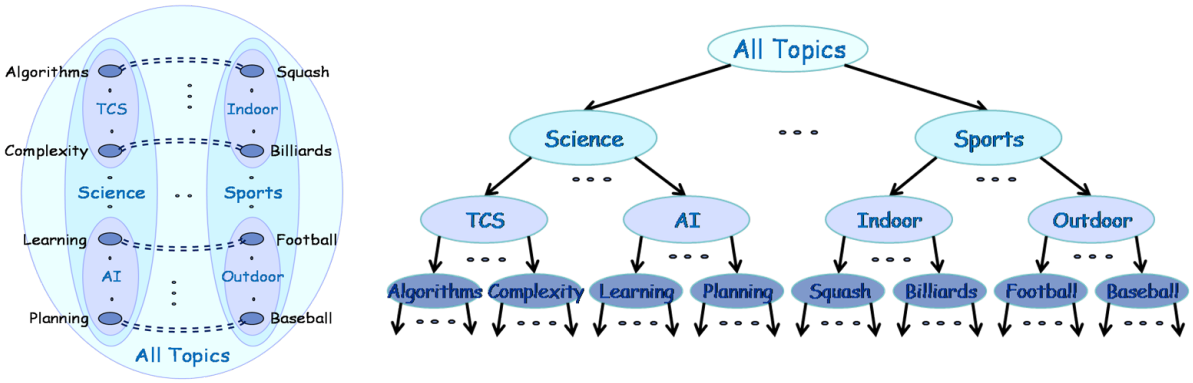


Figure 17: Same as Figure 16 except let us match each point in Algorithms with a point in Squash, each point in Complexity with a point in Billiards, each point in Learning with a point in Football, and each point in Planning with a point in region Baseball. Define the similarity measure to be the same as in Figure 16 except that we let $\mathcal{K}(x, y) = 1$ if x and y are matched. Note that for $\alpha = 1/n$ the similarity function satisfies the α-good neighborhood with respect to any of the prunings of the tree above. However, standard linkage algorithms would initially link the matched pairs and produce clusters with very high error with respect to any such clustering.

In particular, single linkage, complete linkage, average linkage, centroid linkage and median linkage would in the first $n/2$ stages merge the matched pairs of points. From that moment on, no matter how they perform, none of the natural and desired clusterings will even be $1/2$ close to any of the prunings of the hierarchy produced. Notice however, that $\mathcal{K}$ satisfies α-good neighborhood with respect to any of the desired clusterings (for $\alpha = 1/n$), and that our algorithm will be successful on this instance. The ν-strict separation is not satisfied in this example either, for any constant ν.

[2]Since, usually, the similarity function between pairs of objects is constructed based on heuristics, this can easily happen; for example we could have a similarity measure that puts a lot of weight on features such as date or names, and so we could easily have a document about Learning being more similar to a document about Football than to other documents about Learning.

3.5 Robustness of Algorithms w.r.t. Structure of Data

Most of the standard linkage methods do not work well in the presence of non spherical clusters or outliers. Single Linkage is better at clustering arbitrary shapes, but is very sensitive to outliers. It merges the two ellipsoids because it cannot handle the chain of outliers connecting them. (Figures 19 and 20). In cases where cluster sizes vary, complete linkage and average linkage break larger clusters into parts.

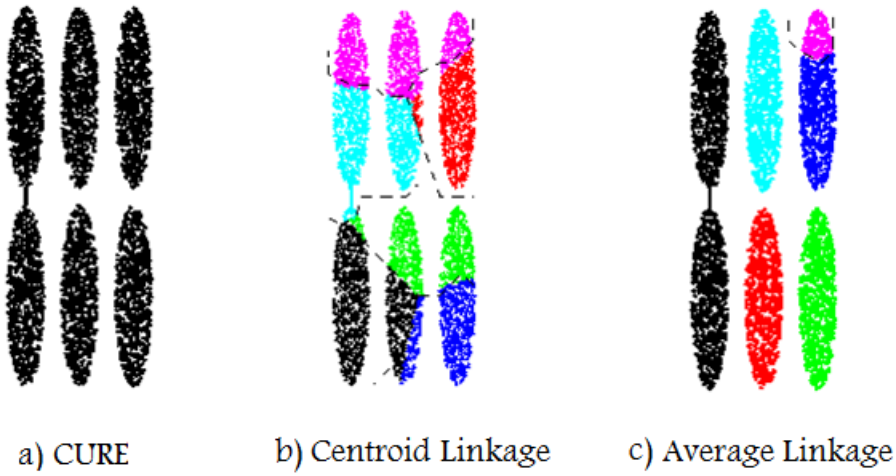


Figure 18: Comparison of *CURE* and Standard Linkage Algorithms when clusters are non-spherical with presence of outliers.

Centroid linkage, median linkage and ward's linkage are defined only for data in metric space and assumes that input data gives the distance from origin. A major disadvantage of centroid linkage is that if the sizes of the two groups to be merged are quite different, then the centroid of the new group will be very close to that of the larger group and may remain within that group (Everitt, 1993). Median method is not suitable for measures such as correlation coefficients, since interpretation in a geometrical sense is no longer possible.

For the centroid-based algorithm, the space that constitutes the vicinity of the single centroid for a cluster is spherical. Thus, it favors spherical clusters and splits the elongated clusters (as shown in Figure 18 and 19).

The scattered points approach employed by *CURE* alleviates the shortcomings of both

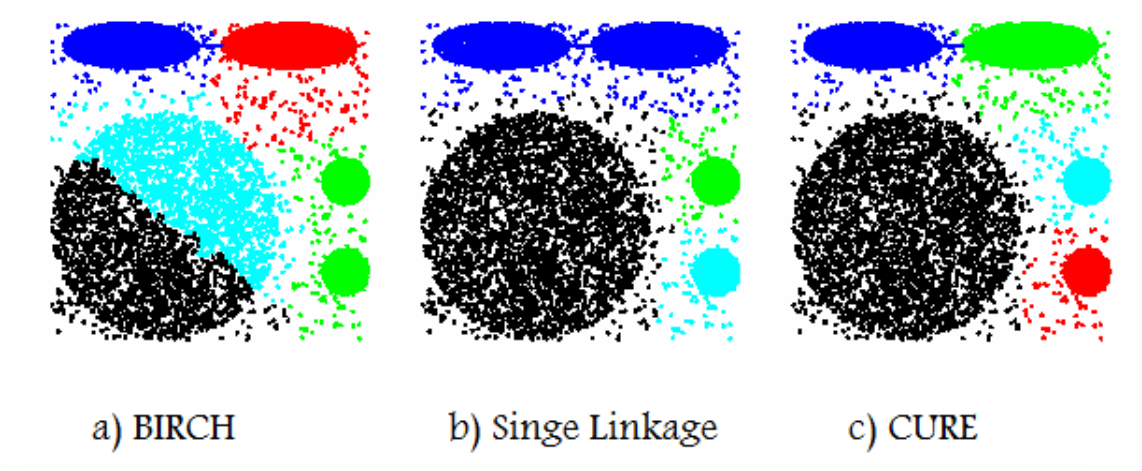


Figure 19: Comparison of *CURE*, *BIRCH* and Single Linkage when some clusters are non-spherical and of varying sizes with presence of outliers.

the all-points (*e.g.* single, complete, average linkage) as well as the centroid-based linkage approaches (*e.g.* centroid, median linkage). It enables *CURE* to correctly identify the clusters(Figures 18, 19, 20). *CURE* is less sensitive to outliers since shrinking the scattered points toward the mean dampens the adverse effects due to outliers as outliers are typically further away from the mean and are thus shifted a larger distance due to the shrinking (Figures 18, 19, 20). Multiple scattered points also enable *CURE* to discover non-spherical clusters like the elongated clusters (as shown in Figures 18, 19 and 20).

On the other hand, since *BIRCH* uses a centroid-based hierarchical clustering algorithm for clustering the preclustered points, it cannot distinguish between the big and small clusters. In Figure 19 it splits the larger cluster while merging the two smaller clusters adjacent to it. Moreover, it fails to identify clusters with non-spherical shapes (e.g., elongated) (Figures 19 and 20).

However, *CURE* fails when data has clusters with different densities as shown in Figure 21. Moreover, though *CURE* does overcome some noise in the data, still it fail in case of *sparse noise* (a large fraction of the points in the data have very high similarity to a few points from other target clusters even though most of the nearest neighbors are from their own target cluster).

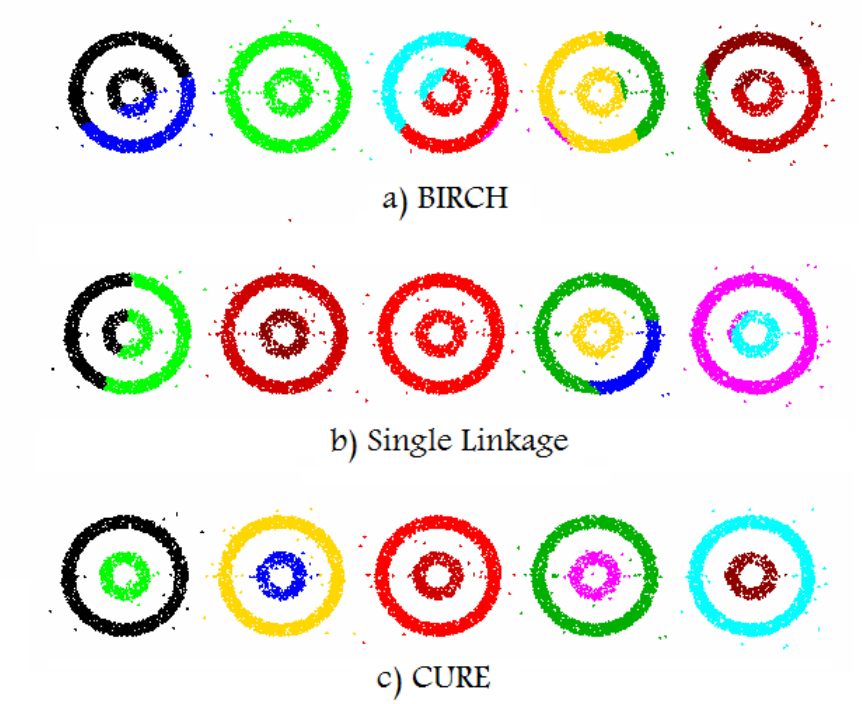
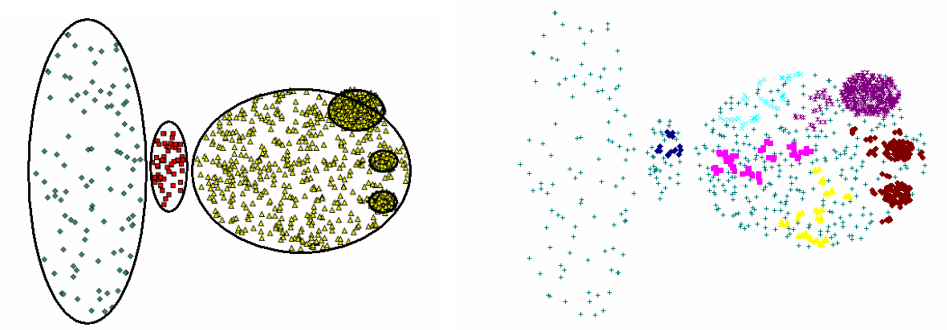


Figure 20: Comparison of *CURE*, *BIRCH* and Single Linkage when clusters are of varying sizes with presence of outliers.



(a) Target Clusters (b) Clusters identified by *CURE*

Figure 21: *CURE* fails when clusters are of different density.

In general, hierarchical algorithms can be identified based on whether they merge two clusters based on *closeness*, *e.g.* single linkage, complete linkage, *CURE*, etc. or whether they merge clusters based on *average connectivity*, *e.g.* average linkage, etc. In Figure 22, all *closeness* schemes would merge clusters (*a*) and (*b*) whereas all *average connectivity*

schemes would merge clusters (c) and (d).

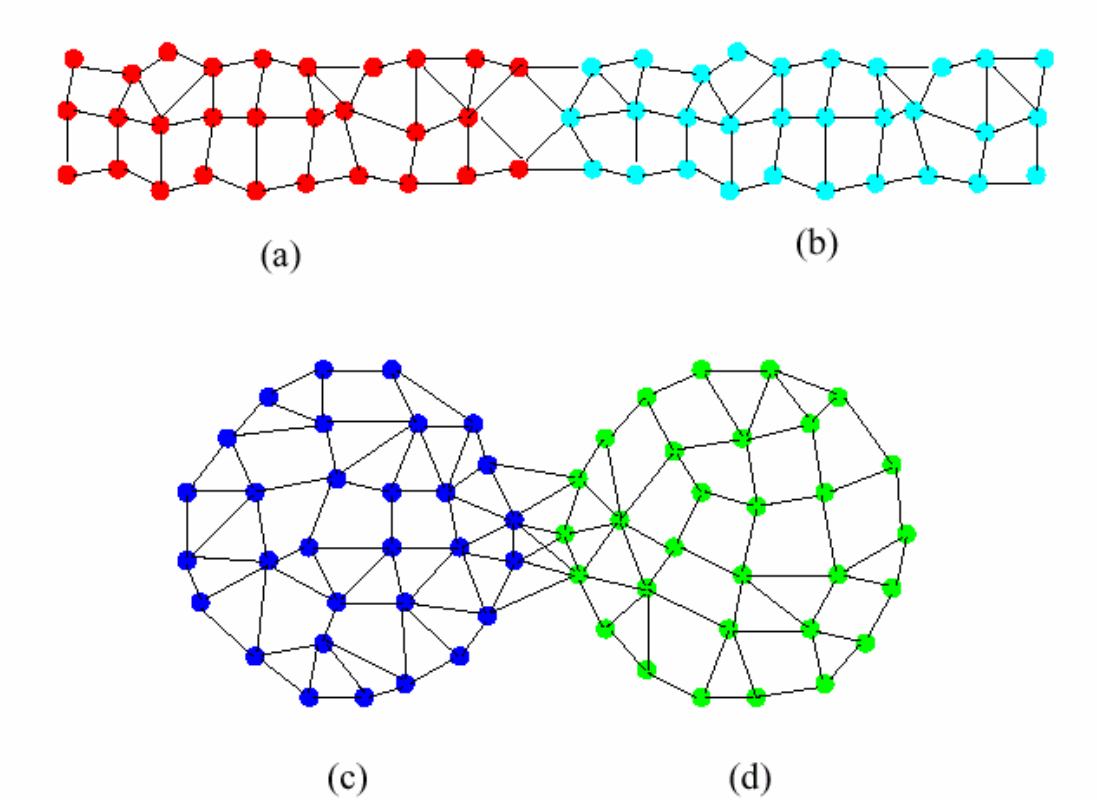


Figure 22: Comparison of *closeness* schemes and *average connectivity* schemes. (a), (b), (c) and (d) are target clusters.

3.6 Discussion

In this chapter we analyzed the performance of different hierarchical algorithms and looked at both theoretical and experimental results in order to understand how different algorithms behave under different conditions such as structure of data, presence of noise, incompleteness in data, etc.

We described the framework defined by (Balcan et al., 2008) that we use throughout our work to evaluate the correctness and robustness of algorithms. Under this framework we described several interesting properties (like *strict separation, stability*, etc.) of similarity function and discussed some of the correctness results for linkage algorithms when data satisfies these natural properties.

We further proposed two new natural properties of data called *good neighborhood* properties that capture extremely well various forms of real world noise like presence of outliers, incompleteness of data, data corruption, transmission errors, etc. In these properties, all points in the data set are allowed to be affected by some amount of noise. We then gave examples that even though the data may satisfy these *good neighborhood* properties, most standard linkage algorithms fail miserably.

We discussed through examples the robustness of different hierarchical algorithms with respect to different structural properties of data like shape, size and density of target clusters in the data set. We saw that though an algorithm may be robust to some of the structural properties, however it could fail miserably under other conditions. *e.g.* $CURE$ is not only able to handle clusters of non-spherical shape but can also cluster well when sizes of the target clusters are vastly different. However, it fails if target clusters have varying densities. Moreover, though $CURE$ does overcome some noise (*e.g.* outliers) in the data, still it fail in case of other forms of noise (*e.g. sparse noise*).

CHAPTER IV

ROBUST HIERARCHICAL LINKAGE (*RHL*)

In this chapter we propose a new robust algorithm and prove that is successful if the data is noisy, *i.e.* if the data satisfies the good neighborhood property. This procedure has two phases:

1^{st} ***Phase*:** It uses somewhat more global information for creating an interesting starting point for a linkage procedure – a set of not too small, but also not too large blobs that are mostly "pure".

2^{nd} ***Phase*:** It then runs robust linkage procedure, called *Ranked Linkage*, on this set of blobs.

Use of blobs and statistical median similarity information lends robustness since noisy similarities are outvoted during merging. Both steps have to be done with care and we will describe in detail in the following sections both steps of our algorithm. In particular, in Section 4.1 we describe the procedure for generating a set of interesting blobs and in Section 4.2 we describe the linkage procedure.

Note that (Balcan et al., 2008) have shown that if $\mathcal{K}$ satisfies the strict separation property with respect to the target clustering, then as long as the smallest target cluster has size $5\nu n$, one can in polynomial time construct a hierarchy such that the ground-truth is ν-close to a pruning of the hierarchy. Unfortunately, the algorithm presented in (Balcan et al., 2008) is computationally very expensive:

- It first generate a large list of $\Omega(n^2)$ candidate clusters.
- Then it repeatedly runs pairwise tests in order to laminarize these clusters.

Thus, it's running time is a large unspecified polynomial. On the other hand, our new robust linkage algorithm can be used to get a simpler and much faster algorithm for clustering accurately under the ν-strict separation property. Additionally, our algorithm is much more general as well as it provably works under the more general (α, ν)-good neighborhood property.

Before describing the algorithm, we introduce some notation used throughout this chapter.

Notation:

For the rest of this chapter we assume that the similarity function $\mathcal{K}$ satisfies the (α, ν)-good neighborhood property for the clustering problem (S, ℓ). Let $S' \subseteq S$ be the set of $(1-\nu)n$ points such that $\mathcal{K}$ satisfies α-good neighborhood with respect to S'. We call the points in S' good points and the points in $S \setminus S'$ bad points. Let $G_i = C_i \cap S'$ be the good set of label i. Let $G = \cup G_i$ the whole set of good points; so $G = S'$. Clearly $|G| \geq n - \nu n$. Denote by C_G the restriction of the target clustering to the set G.

Definition 1. *For $A \subseteq S$, $B \subseteq S$ we define*

$$\mathcal{K}_{\text{median}}(A, B) = \text{median}\{\mathcal{K}(x, x'); x \in A, x' \in B\} \tag{18}$$

and we call this the median similarity *of A to B.*

For simplicity we denote $\mathcal{K}_{\text{median}}(\{x\}, B)$ as $\mathcal{K}_{\text{median}}(x, B)$.

4.1 Generating an Interesting Starting Point

In this section we describe the first phase of the algorithm in which we generate a list L of blobs using the (α, ν)-good neighborhood property.

For Algorithm 6, we can show the following:

Theorem 5. *Let $\mathcal{K}$ be a symmetric similarity function satisfying the (α, ν)-good neighborhood for the clustering problem (S, ℓ). So long as the smallest target cluster has size*

Algorithm 6 Generate Interesting Blobs

Input: similarity function $\mathcal{K}$, set of points S, $\nu > 0$, $\alpha > 0$.

Let the initial threshold $t = (5\alpha + 4\nu)n + 1$. Let L be empty. Let $A_S = S$.

Step 1 Construct the graph F_t where we connect points x and y in A_S if they share $t - (\nu + 2\alpha)n - 2$ points in common out of their t nearest neighbors with respect to the whole set of points S.

Step 2 Construct the graph H_t by connecting points x and y if they share at least νn neighbors in the graph F_t.

Step 3 (i) Add to L all the components C of H_t with $|C| \geq 2(\nu + \alpha)n + 1$ and remove from A_S all the points in all these components.

(ii) For all points x in A_S check if $(\nu + \alpha)n$ out of their $4(\nu + \alpha)n + 1$ nearest neighbors are in L. If so, then assign point x to any of the blobs in L of highest median. Remove the points in all these components from A_S.

Step 4 While $|A_s| > 2(\nu + \alpha)n$ and $t < n$, increase the critical threshold and go to Step 1.

Step 5 Assign all points x that do not belong to any of the blobs in L arbitrarily to one of the blobs.

Output: A list of blobs which form a partition of S.

greater than $9(\nu + \alpha)n$*, then we can use Algorithm 6 to create a list L of blobs each of size at least* $3(\nu + \alpha)n$ *such that:*

- *The blobs in L form a partition of S.*
- *Each blob in the list L contains good points from only one good set; i.e. for any* $C \in L$, $C \cap G \subseteq G_i$ *for some* $i \leq k$.

Proof: In the following we denote by n_{C_i} the number of points in the target cluster i. Without loss of generality assume that $n_{C_1} \leq n_{C_2} \leq ... \leq n_{C_k}$. We will show by inductions on $i \leq k$ that:

(a) For any $t \leq n_{C_i}$, any blob in the list L only contains good points from a single good set G_i; all blobs have size at least $3(\nu + \alpha)n$.

(b) At the beginning of the iteration $t = n_{C_i} + 1$, any good point $x \in C_j \cap G$, $j \in \{1, 2, \ldots, i\}$

has already been assigned to a blob in the list L that contains points only from $C_j \cap G$ and has more good points than bad points.

These two claims clearly imply that each blob in the list we output contains good points from only one good set. Moreover at $t = n_{C_k}$ all good points have been assigned to one of the blobs in L. Since we assign the remaining points x that do not belong to any of the blobs in L (these can only be bad points) arbitrarily to one of the blobs, we also get that the blobs in L form a partition of S, as desired.

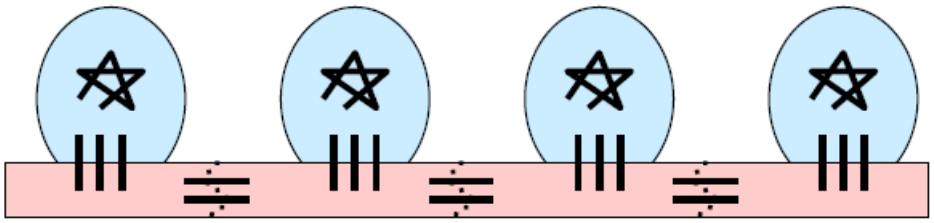


Figure 23: Graph F_t. No good point in cluster i is connected to a good point in a different cluster j, $i \neq j$. No bad point is connected to both a good point in cluster i and a good point in different cluster j, $i \neq j$.

Claims (a) and (b) are clearly both true initially. We show now that as long as $t \leq n_{C_1}$, the graphs F_t and H_t have the following properties:

(1) No good point in cluster i is connected in F_t to a good point in a different cluster j, for $i, j > 1$, $i \neq j$ (Figure 23). This follows from the (α, ν)- good neighborhood property: as long as t is smaller than n_{C_i} for any good point $x \in C_i$ all but at most $(\nu + \alpha)n$ out of it's t nearest neighbors lie in it's good set, *i.e.* $|C_i(x) \cap G|$; similarly, as long as t is smaller than n_{C_j} for any good point $y \in C_j$, all but at most $(\nu + \alpha)n$ out of it's t nearest neighbors lie in it's good set, *i.e.* $|C_j(x) \cap G|$; so it cannot be the case that for $6(\nu + \alpha)n \leq t \leq n_{C_1}$ two good points in two different clusters $i, j \geq 1$ share $t - 2(\nu + \alpha)n$ points in common out of their t nearest neighbors.

(2) No bad point is connected in F_t to both a good point in cluster i and a good point in different cluster j, for $i, j > 1$, $i \neq j$ (Figure 23). This again follows from the fact that

since $t \leq n_{C_i}$ for all i, for any good point x all but at most $(\nu+\alpha)n$ out of it's t nearest neighbors lie in it's good set $|C(x) \cap S|$; so for a bad point z to share $t-2(\nu+\alpha)n$ points out of it's t nearest neighbors in common with the t nearest neighbors of a good point x in G_i it must be the case that z has $t-3(\nu+\alpha)n$ points out of it's t nearest neighbors in G_i; but that means that there cannot be another good point y in G_j, where $j \neq i$ such that z and y share $t-2(\nu+\alpha)n$ points among their t nearest neighbors, because we would need to have that $t-3(\nu+\alpha)n$ of points out of z's t nearest neighbors lie in G_i and $t-3(\nu+\alpha)n$ of points out of z's t nearest neighbors in G_j; but this cannot happen if $t > 6(\nu+\alpha)n$ since $2(t-3(\nu+\alpha)n) > t$ for $t > 6(\nu+\alpha)n$.

(3) All the components of H_t of size at least $3(\nu+\alpha)n$ will only contain good points from one cluster (Figure 23). Since in F_t bad points can only connect to one good set, we get that no two good points in the different clusters connect in H_t.

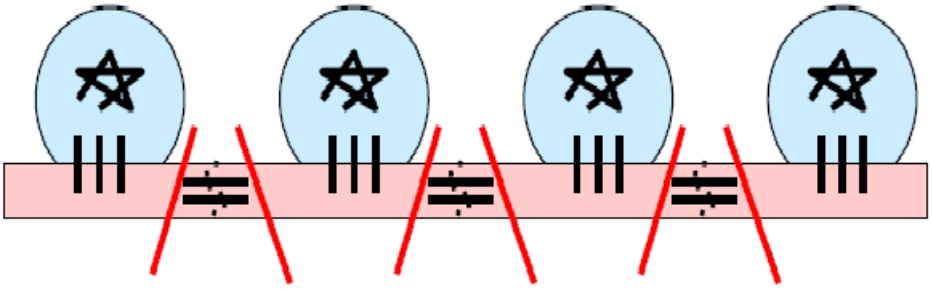


Figure 24: Graph H_t. All the components of H_t of size at least $3(\nu+\alpha)n$ contain good points from only one cluster.

We can use (1), (2), and (3) to argue that as long as $t \leq n_{C_1}$, each blob in L contains good points from at most one target cluster. This is true at the beginning and by (3), for any $t \leq n_{C_1}$, anytime we insert a whole new blob in L in Step 3(i), that blob must contain good points from at most one target cluster.

We now argue that this property is never violated as we assign points to blobs already in L based on the median test in Step 3(ii). Note that at all time steps all the blobs in L have size at least $3(\nu+\alpha)n$. Assume that a good point x has more than $(\nu+\alpha)n$ out of it's

$5(\nu+\alpha)n$ nearest neighbors in S in the list L. By Lemma 2, there must exist a blob in L that contains only good points from $C(x)$. By Lemma 1, if we assign x based on the median test in Step 3(ii), then we will add x to a blob containing good points only from $C(x)$, and so we maintain the invariant that each blob in L contains good points from at most one target cluster.

We now show that at the beginning of the iteration $t = n_{C_1} + 1$, all the good points in C_1 have already been assigned to a blob in the list L that only contains good points from $C_1 \cap G$. There are a few cases.

First, if prior to $t = n_{C_1}$ we did not yet extract in the list L a blob with good points from C_1, then it must be the case that all good points in C_1 connect to each other in the graph F_t; so there will be a component of H_t that will contain all good points from C_1 and potentially bad points, but no good points from another target cluster; moreover this $|C_1| \geq 9(\nu + \alpha)n$, this component will be output in Step 3(i).

Second, if prior to $t = n_{C_1}$ we did extract some, but still, more than $3(\nu+\alpha)n$ points from the good set G_1 do not belong to blobs in the list L, then more than $3(\nu+\alpha)n$ of good points will connect to each other in F_t, and then in H_t, so we will add one blob to L containing these good points (plus at most νn bad points).

Finally, it could be that by the time we reach $t = n_{C_1}$ all but $l < 3(\nu + \alpha)n$ good points in C_1 have been assigned to a blob in the list L that has good points only from C_1. Since $|C_1| \geq 9(\nu + \alpha)n$ we must have assigned at least $9(\nu + \alpha)n - 3(\nu + \alpha) - \nu n \geq 5(\nu + \alpha)n$ good points from C_1 to the list L. This together with the (α, ν)-good neighborhood property implies that the good points in C_1 that do not belong to the list L yet, must have $(\nu+\alpha)n$ out of their $5(\nu + \alpha)n$ nearest neighbors in S in the list L (at most ν out of the $5(\nu + \alpha)n$ nearest neighbors can be bad points, at most αn can be good points from a different cluster, and at most $3(\nu + \alpha)n$ can be good points in C_1 that do not yet belong to L).

So we will assign these points to blobs in L based on the median test in Step 3(ii). By Lemma 1, when we assign them based on the median test in Step 3(ii), we will add them to

a blob containing good points from C_1 and no good points from other cluster C_j, as desired.

We then iterate the argument on the remaining set A_S. The key point is that for $t \geq n_i$, $i > 1$, once we start analyzing good points in C_{n_i+1} we have that *all* the good points in C_{n_i}, $C_{n_{i-1}}$, ..., C_{n_1} have already been assigned to blobs in L. ■

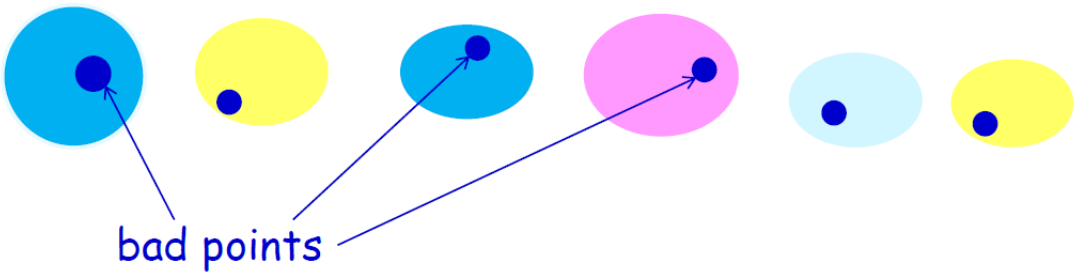


Figure 25: List L of disjoint clusters each of size at least $3(\nu + \alpha)n$ where each cluster in L intersects at most one good set.

Lemma 1. *Let $\mathcal{K}$ be a symmetric similarity function satisfying the (α, ν)-good neighborhood for the clustering problem (S, ℓ). Assume that L is a list of disjoint clusters each of size at least $3(\nu + \alpha)n$. Assume also that each cluster in L intersects at most one good set; i.e. for any C in L, we have $C \cap G \subseteq G_i$ for some i. Consider $x \in G$ such that there exist C in L with $C \cap G \subseteq C(x) \cap G$. Let $\tilde{C}$ be the blob in L of highest median similarity to x. Then $\tilde{C} \cap G \subseteq C(x) \cap G$.*

Proof: Let us fix a good point x. Let C' and C'' be such that $C' \cap G \subseteq C(x) \cap G$ and $C'' \cap G \subseteq C_i \cap G$, for $C_i \neq C(x)$. Since $\mathcal{K}$ is a symmetric similarity function satisfying the (α, ν)-good neighborhood property we have that x can be more similar to at most $\nu n + \alpha n$ points in C'' than with any point in $C' \cap G$. Since $|C'| \geq 3(\nu + \alpha)n$ and $|C''| \geq 3(\nu + \alpha)n$ we get that $\mathcal{K}_{\text{median}}(x, C') \geq \mathcal{K}_{\text{median}}(x, C'')$. This then implies that the blob $\tilde{C}$ in L of highest median similarity to x must satisfy $\tilde{C} \cap G \subseteq C(x) \cap G$, as desired. ■

Lemma 2. *Let $\mathcal{K}$ be a symmetric similarity function satisfying the (α, ν)-good neighborhood for the clustering problem (S, ℓ). Assume that L is a list of clusters each of size at least $3(\nu + \alpha)n$. Assume also that each cluster in L intersects at most a good set; i.e. for*

any C in L, we have $C \cap G \subseteq G_i$ for some i. Consider $x \in G$ such that there is no cluster C in L with $C \cap G \subseteq C(x) \cap G$. Then at most $(\alpha + \nu)n$ of it's t nearest neighbors for any $t \leq n_{C(x)}$ can be in L and all the rest are outside.

Proof: By the (α, ν)-good neighborhood for all $x \in G$, for all $t \leq n_{C(x)}$ at most $(\alpha + \nu)n$ of it's t nearest neighbors are not from $C(x)$. Consider $x \in G$ such that there is no cluster C in L with $C \cap G \subseteq C(x) \cap G$. So, the list L contains no points from C, which implies that at most $(\alpha + \nu)n$ of it's t nearest neighbors for any $t \leq n_{C(x)}$ can be in L. ■

4.2 Ranked Linkage

In this section we describe a new robust linkage procedure, *Ranked Linkage*, that uses the a list L of blobs generated by Algorithm 6 and generates a hierarchy using a symmetric score function as a measure of similarity between clusters. The linkage procedure is described in Algorithm 7.

Algorithm 7 Ranked Linkage

Input: A list L of blobs; similarity function $\mathcal{K}$ on pairs of points.

- Repeat till only one cluster remains in L:
  - (a) Find clusters C, C' in the current list which maximize score(C, C')
  - (b) remove C and C' from L merge them into a single cluster and add that cluster to L.
- Let T be the tree with single elements as leaves and internal nodes corresponding to all the merges performed.

Output: Tree T on subsets of S.

We describe in the following the notion of similarity between pairs of blobs used in Algorithm 7.

Definition 2. *Let $L = \{A_1, \ldots, A_l\}$ be a list of disjoint subsets of S. For each i, for each point x in A_i we compute $\mathcal{K}_{\text{median}}(\{x\}, A_j)$, $j \neq i$, sort them in increasing order, and define*

$\text{rank}(x, A_j)$ *as the rank of* A_j *in the order induced by* x*. We define*

$$\text{rank}(A_i, A_j) = \text{median}_{x \in A_i}[\text{rank}(x, A_j)].$$

For example, if A_{j_1} is the subset of highest median similarity to x out of all A_j, $j \neq i$, then $\text{rank}(x, A_{j_1}) = l$. Similarly, if A_{j_2} is the subset of smallest median similarity to x out of all A_j, $j \neq i$, then $\text{rank}(x, A_{j_2}) = 1$.

Definition 3. *Let* $L = \{A_1, \ldots, A_l\}$ *be a list of disjoint subsets of* S*. We define the score between* A_i *and* A_j *as*

$$\text{score}(A_i, A_j) = \min[\text{rank}(A_i, A_j), \text{rank}(A_j, A_i)].$$

Note: While $\text{rank}(\cdot, \cdot)$ might be asymmetric, $\text{score}(\cdot, \cdot)$ is designed to be symmetric.

We now present a useful lemma.

Lemma 3. *Let* $\mathcal{K}$ *be a symmetric similarity function satisfying the* (α, ν)*-good neighborhood for the clustering problem* (S, ℓ)*. Let* L *be a list of disjoint clusters, all of size at least* $2(\alpha + \nu)n$*. Assume that* $B \cap G \subseteq G_i$*,* $B' \cap G \subseteq G_i$*, and* $(B'' \cap G) \cap G_i = \emptyset$*. Then we have both*

$$\text{score}(B, B') < \text{score}(B, B'') \quad \text{and} \quad \text{score}(B, B') < \text{score}(B', B'').$$

Proof: Let x be a good point. The (α, ν)-good neighborhood property implies that there exists c_x such that at most αn points $z \in G$, $z \notin C(x)$ can have similarity $K(x, z)$ greater or equal to c_x and at most αn points $y \in G \cap C(x)$ can have similarity $K(x, z)$ strictly smaller than c_x. Since each of the blobs has size at least $2(\alpha + \nu)n$ and since each blob contains at most νn bad points, we get that for all blobs B' and B'' such that $B' \cap G \subseteq C(x) \cap G$ and $(B'' \cap G) \cap (C(x) \cap G) = \emptyset$ we have

$$\mathcal{K}_{\text{median}}(\{x\}, B') > \mathcal{K}_{\text{median}}(\{x\}, B'').$$

So a good point x will rank blobs B' s.t. $B' \cap G \subseteq C(x) \cap G$ later than blobs B'' such that $(B'' \cap G) \cap (C(x) \cap G) = \emptyset$ in the order it induces. Assume that there are exactly r blobs B in

L such that $(B \cap G) \cap (C(x) \cap G) = \emptyset$. Since there are at most νn bad points and each of the blobs has size at least $2(\alpha + \nu)n$, we obtain that for all B, B' in L such that $B \cap G \subseteq C_i \cap G$ and $B' \cap G \subseteq C_i \cap G$, and for all B'' in L with $(B'' \cap G) \cap (C_i \cap G) = \emptyset$ we have both

$$\text{rank}(B, B') > r \geq \text{rank}(B, B'') \quad \text{and} \quad \text{rank}(B', B) \geq r \geq \text{rank}(B', B'').$$

This then implies that

$$\text{score}(B, B') = \text{score}(B', B) = \min[\text{rank}(B, B'), \text{rank}(B', B)] > r.$$

Similarly,

$$\text{score}(B, B'') = \text{score}(B'', B) = \min[\text{rank}(B, B''), \text{rank}(B'', B)] > r.$$

Finally, we have

$$\text{score}(B', B'') = \text{score}(B'', B') = \min[\text{rank}(B', B''), \text{rank}(B'', B')] \leq r.$$

These imply:

$$\text{score}(B, B') > \text{score}(B, B'') \quad \text{and} \quad \text{score}(B, B') > \text{score}(B', B''),$$

as desired. ■

We now show that is the similarity function we have satisfies the good neighborhood property, given a good starting point, Algorithm 7 will be successful in outputting a good hierarchy.

Theorem 6. *Let $\mathcal{K}$ be a symmetric similarity function satisfying the (α, ν)-good neighborhood for the clustering problem (S, ℓ). Assume that L is a list of clusters each of size at least $3(\nu + \alpha)n$ that partition the entire set of points. Assume also that each cluster in L intersects at most a good set; i.e. for any C in L, we have $C \cap G \subseteq G_i$ for some i. Then Algorithm 7 constructs a binary tree such that the ground-truth clustering is ν-close to a pruning of this tree.*

Proof: First note that at each moment the list L of clusters is a partition of the whole dataset and that all clusters in L have size at least $3(\nu + \alpha)n$. We prove by induction that at each time step the list of clusters restricted to G is laminar w.r.t. $\mathcal{C}_G$.

In particular, assume that our current list of clusters restricted to G is laminar with respect to $\mathcal{C}_G$ (which is true at the start). This implies that for each cluster C in our current clustering and each C_r in the ground truth, we have either

$$C \cap G \subseteq G(C_r) \quad \text{or} \quad G(C_r) \subseteq C \cap G \quad \text{or} \quad (C \cap G) \cap G(C_r) = \emptyset.$$

Now, consider a merge of two clusters C and C'. The only way that laminarity could fail to be satisfied after the merge is if for one of the two clusters, say, C', we have that $C' \cap G$ is strictly contained inside $C_{r'} \cap G$, for some ground-truth cluster $C_{r'}$ (so, $(C_{r'} \cap G) \setminus (C' \cap G) \neq \emptyset$, $(C' \cap G) \subset C_{r'}$) and yet $C \cap G$ is disjoint from $C_{r'} \cap G$. But there must exist C'' in the list L such that $(C'' \cap G) \subset C_{r'} \setminus (C' \cap G)$, $|C''| \geq 3(\nu + \alpha)n$. By Lemma 3 we know that

$$\text{score}(C', C'') > \text{score}(C', C).$$

However, this contradicts the specification of the algorithm, since by definition it merges the pair C, C' such that score(C', C) is greatest. ■

4.3 The Main Result

Here we present the main algorithm combining Algorithm 6 and Algorithm 7.

Algorithm 8 Robust Hierarchical Linkage (*RHL*)

Input: Similarity function $\mathcal{K}$, set of points S, $\nu > 0$, $\alpha > 0$.

1. Run Algorithm 6 with parameters ν, α to generate an interesting list L of blobs that partitions the whole set S.
2. Run the Ranked Linkage Algorithm 7 on these blobs to get the tree T.

Output: Tree T on subsets of S.

Theorem 7. *Let $\mathcal{K}$ be a symmetric similarity function satisfying the (α, ν)-good neighborhood for the clustering problem (S, ℓ) As long as the smallest target cluster has size*

greater than $9(\nu+\alpha)n$*, then we can use Algorithm 8 in order to produce a tree such that the ground-truth clustering is* ν*-close to a pruning of this tree in* $O(n^{\omega+1})$ *time, where* $O(n^{\omega})$ *is the state of the art for matrix multiplication.*

Proof: The correctness follows immediately from Theorems 5 and 6. The running time follows from Theorem 8 in Section 4.4. ■

4.4 Run Time Analysis

In this section we present an analysis for the running time of Algorithm 8.

Theorem 8. *Algorithm 8 has a running time of* $O(n^{\omega+1})$*, where* $O(n^{\omega})$ *is the state of the art for matrix multiplication. (The current value of* ω *is 2.376 (Coppersmith & Winograd, 1987)).*

Proof: We analyze the running times of Algorithms 6 and 7 separately. For Algorithm 6, we also discuss certain data structures which are utilized throughout the algorithm for speed up.

In the preprocessing step, we construct a list of nearest neighbors for each point in S by sorting $n-1$ other points in decreasing order of similarity. This takes $\mathrm{O}(n \log n)$ time for each point and thus the entire preprocessing step costs $\mathrm{O}(n^2 \log n)$ time.

Now, think of a directed t-regular graph E_t, where, for each point j in the t nearest neighbors of a point i, there is a directed edge from i to j in E_t. We construct two n x n matrices Adj^E and $Nbrs^E$. Adj^E is the adjacency matrix for the graph E_t and $Nbrs^E = Adj^E \mathrm{x} (Adj^E)^T$. $Nbrs^E_{ij}$ gives the number of common neighbors between i and j in E_t and from this, we can know whether to draw an edge between them in F_t. Constructing $Nbrs^E$ for the first time takes $O(n^{\omega})$ time. Notice, however, we do not require to recompute $Nbrs^E$ from scratch in every iteration over t as the graph E_t is monotonically increasing. In iteration $t+1$, for each point i exactly one new edge is added to another point k in the graph E_t where k is the $(t+1)_{th}$ nearest neighbor of i. If there was already

an edge from k to i in E_t, the value $Nbrs^E_{ik}$ (and $Nbrs^E_{ki}$) increases by 1. For every point j that has a directed edge to k, $Nbrs^E_{ij}$ (and $Nbrs^E_{ji}$) increases by 1. This can be computed by comparing the row Adj^E_i of Adj^E with column Adj^E_k of Adj^E. This requires $O(n)$ time for each point i. Since there are n such points the total cost of the iteration is $O(n^2)$. There can be a maximum of $O(n)$ such iterations. Hence, the total cost of updating $Nbrs^E$ over all iterations is $O(n^3)$. Now, let us consider the case when a new blob of size at least $3(\nu + \alpha)n$ is found and added to L. We remove this blob from A_s and correspondingly all the points in the blob from Adj^E and Adj^E. For each such removal we need to recompute Adj^E and $Nbrs^E$. There can be at most $n/3(\nu + \alpha)n = 1/3(\nu + \alpha)$ such iterations. Thus, we take at most $O(n^\omega/(\nu + \alpha))$ time. Therefore, the total cost of constructing the graph F_t over all iterations is $O(n^\omega(1/(\nu + \alpha) + n^{3-\omega}))$.

Similarly, let us have two n x n matrices Adj^F and $Nbrs^F$, where Adj^F is the adjacency matrix of the undirected graph F_t and $Nbrs^F = Adj^F \text{x} Adj^F$ (notice, F_t is undirected). Note that the same trick does not hold while constructing the graph H_{t+1} from H_t as the graph F_t is not monotonically increasing. e.g. two points x and y which had an edge in F_t as they had exactly $t - 2(\nu + \alpha)n$ neighbors in common may not have an edge in F_{t+1} any more as they might not have $t + 1 - 2(\nu + \alpha)n$ neighbors in common. Thus, for each iteration we need to recompute the matrix $Nbrs^F$. Thus, to construct H_t, it takes a total of $O(n^{\omega+1})$ time over all iterations.

In Step 3(i), we can find all the components of H_t in at most O($|V_{H_t}| + |E_{H_t}|$) time where $|V_{H_t}| = n$ and $|E_{H_t}| = O(n^2)$. Thus, we can do this in at most O(n^2) time. Now let's look at the Step 3(ii). It is easy to see that we do this Step at most $1/3(\nu + \alpha)$ times, *i.e.* once for each new blob. For the first stage of this Step, we maintain a set P_i for each point i that is a set of points from it's first $5(\nu + \alpha)$ neighbors that are not yet in the list L and a count c_i of the points already present in L. Every time a new blob B is added to L or we get a non-empty set T (construction explained below) after an iteration of Step 3(ii), we check for the common points between P_i and B or T, then remove all such points from P_i and

add the number of these points to c_i. Since, there can be at most $O(n)$ such iterations, this step can take $O(n^3)$ time over all iterations for all points. For each point x that makes it to the next stage of Step 3(ii), we compute the median to each blob B present in L. This can be done in $O(|B|)$ time for each blob (Blum et al., 1972). For all the blobs, this can be done in $\sum_{B_i \in L} O(|B_i|) \leq O(n)$ time. Once we have the median for all blobs, we can find the highest of these ($\#blobs \leq 1/3(\nu + \alpha)$) medians which requires at most $O(1/(\nu + \alpha))$ time. Notice, that once a point x gets to this stage, it will be added to one of the blobs in L and not be considered again, which means we hit this stage at most $O(n)$ times. Thus, over all iterations, for all points, this stage of Step 3(ii) can be done in $O(n^2)$ time. Notice, we need to do an additional post-processing step. We add all such x to a set T. This set T is compared with P_i (as explained above) at the beginning of the next iteration. The addition of these points to L could result in more points satisfying the first condition of Step 3(ii). Therefore, the time required for Step 3 is $O(n^3)$.

The costliest Steps in the Algorithm 6 is Step 2 and causes the algorithm to have $O(n^{\omega+1})$ running time.

Algorithm 7 has a running time of $O(n/(\nu + \alpha)(n + 1/(\nu + \alpha)\log(1/(\nu + \alpha))))$. At each level, we compute the score of each cluster with respect to all other clusters. Firstly, we compute the median similarity of each cluster C_i with respect to a point x in $\mathrm{O}(|C_i|)$ time (Blum et al., 1972). This is done for all clusters except $C(x)$. The total running time is $\sum_{C_i \in L: C_i \neq C(x)} O(|C_i|) < O(n)$. Secondly, we compute the rank by sorting these similarity values in ascending order. Since there can be at most $1/3(\nu + \alpha)$ clusters this can be done in $O(1/(\nu + \alpha)\log 1/(\nu + \alpha))$ time. This is done for all points, hence the total cost of computing ranks of all clusters with respect to all points is $O(n(n+1/(\nu+\alpha)\log(1/(\nu+\alpha))))$. Finally, we compute the rank for a cluster C_i with respect to the cluster C_j by taking the median of the ranks given by each point $x \in C_j$ to the cluster C_i and this takes $\mathrm{O}(|C_j|)$ time. This is done by C_j for all other clusters. Thus, the total time taken by C_j to compute ranks for all other clusters is $O(|C_j|/(\nu + \alpha))$. Since we do it for all clusters it requires

$\sum_{C_j \in L} O(|C_j|/(\nu + \alpha)) \leq O(n/(\nu + \alpha))$ time. Thus, we can be done with each level in $O(n(n + 1/(\nu + \alpha)\log(1/(\nu + \alpha))))$ time. The number of levels is at most the number of clusters which is $O(1/(\nu + \alpha))$. Therefore, the total running time of the algorithm is $O(n/(\nu + \alpha)(n + 1/(\nu + \alpha)\log(1/(\nu + \alpha))))$.

For Algorithm 8, the costliest step is Algorithm 6 and thus has a running time of $O(n^{\omega+1})$.

■

4.5 Discussion

In this chapter we proposed and analyzed a new robust algorithm for bottom-up agglomerative clustering. We proved that our algorithm can be used to cluster accurately in cases where the data satisfies a number of natural properties (like the good neighborhood properties which are much more general that the strict separation properties) and where the traditional agglomerative algorithms fail as we have already seen in Chapter 3.

It is also possible to extend *RHL* to an inductive setting with similar correctness guarantees (to be discussed in Chapter 6), where a small subset of points is randomly chosen from a much larger instance space to generate a hierarchy into which the rest of the points are then inserted resulting in a hierarchy over the entire instance space.

We observe in Chapter 7 that experimentally *RHL* does much better than other agglomerative algorithms when tested to various noisy synthetic and real-world data sets.

However, we note here that while being a robust algorithm, *RHL* is not completely agglomerative and is algorithmically quite complex and computationally intensive due to it's running time of $O(n^{\omega+1})$ (Theorem 8).

CHAPTER V

WEIGHTED NEIGHBORHOOD LINKAGE

In Chapter 4, we discussed a new robust algorithm, Robust Hierarchical Linkage (*RHL*). However, *RHL* is not completely agglomerative and is algorithmically quite complex and computationally intensive due to it's running time of $O(n^{\omega+1})$ (Theorem 8). Moreover, it requires two parameters α, ν as input. For new data sets, where the size of smallest cluster is unknown, estimating these values may be quite difficult. In particular, calculating ν is especially hard as it reflects the number of points that terribly noisy. There may however be indirect ways to estimate parameter α as it reflects the possible amount of noise introduced per point. This can be directly assumed from the reliability of data collection or storage, transmission media, etc. *e.g.* general sampling error assumed while collecting data, assumed transmission errors over an unreliable channel, data corruption in storage medium like disk, etc.

Recognizing the usefulness of good neighborhood property in capturing various forms of noise, we propose a new, more practical algorithm, *Weighted Neighborhood Linkage (WNL)*, in this chapter that uses the α-good neighborhood property for formal robustness guarantees and alleviates the limitations of *RHL* by being completely agglomerative in nature, much simpler, faster and more robust to parameter tuning. We show that if the data satisfies the α-good neighborhood property, then our algorithm can be used to output a hierarchy such that the target clustering is a pruning of this hierarchy.

Our objective is to design a completely hierarchical algorithm that uses global structural properties of the data while merging clusters, specifically, the property that if the two clusters have enough common neighbors, then they should be close to each other. The intuition is that if we incrementally explore the nearest neighbors for each cluster and compare

common neighbors, two clusters that are closer to each other will discover more common neighbors quicker than clusters that are far apart.

Notation:

We first introduce some notation used throughout this chapter.

We define a graph $G = (V, E)$, where each vertex $u \in V$ is a cluster of points from S and $|u|$ denotes the number of points in u. Let $N_t(x)$ be the set of t nearest neighbors of each point $x \in S$ with respect to the whole set of points S. Each point is considered it's first nearest neighbor, *i.e.* $N_1(x) = \{x\}$.

For each $x, y \in S$, we define $I_t(x, y)$ to indicate whether y is one of the t nearest neighbors of x or not, *i.e.*

$$I_t(x, y) = \left\{ \begin{array}{ll} 1 & \text{if } y \in N_t(x) \\ 0 & \text{otherwise} \end{array} \right\}$$

For all $u, v \in V$, we define the weight $wt(u, v)$ of the directed edge from u to v, which reflects the average number of nearest neighbors points in u have in v *i.e.*,

$$wt(u, v) = avg_{x \in u} \sum_{y \in v} I_t(x, y) \tag{19}$$

This gives the weight of a neighbor v with respect to u. [1]

We define $C(u, v)$ as a measure of number of common neighbors between two vertices $u, v \in V$ given by

$$C(u, v) = \sum_{w \in V} min(wt(u, w), wt(v, w))$$

Intuitively, greater the value of $C(u, v)$, closer is the cluster u to v. [2] If t nearest neighbors are explored for $x \in S$, then the highest value of $C(u, v)$ can be t, which reflect that all their neighbors are common and should be in turn very close to each other. This definition of $C(u, v)$ makes it possible to merge any two sub-clusters belonging to the same target cluster

[1] If $e = (u, v) \in E$, then $wt(u, v) = wt(e)$ and the notation can be used interchangeably.

[2] $C(u, v)$ is calculated for all $w \in V$ and not $w \in V \backslash \{u, v\}$.

for a threshold that is sufficiently smaller than the threshold for merging two sub-clusters belonging to different target clusters.

We define a function $d(u, A)$ which gives the *total weight* from a vertex $u \in V$ to a subset $A \subseteq V$.

$$d(u, A) = \sum_{v \in A} wt(u, v) \tag{20}$$

This gives the total weight of all the neighbors of u in A. [3]

For the rest of this section we assume that the similarity function $\mathcal{K}$ satisfies the α–good neighborhood property for the clustering problem (S, ℓ). Note that the following is a useful consequence of the definition.

Claim 1. *Let $\mathcal{K}$ be a similarity function satisfying the α–good neighborhood for the clustering problem (S, ℓ). As long as t is smaller than n_{C_i} for any point $x \in C_i$ all but at most αn out of it's t nearest neighbors lie in $C_i(x)$.*

5.1 Algorithm

We design Algorithm 9 such that as the number of explored nearest neighbors (threshold t) reaches a particular value, if there was a target cluster of size equal to this threshold, we should be able to create a vertex in the graph that represented the target cluster, thereby generating all the possible target clusters for the given data set. We formalize this in Theorem 11 which gives the correctness and run time guarantees for the algorithm.

5.2 Correctness Analysis

First, we verify that the total weight of edges to any subset $A \subseteq V$ does not increase if we merge two vertices in Step 3 of the algorithm. This is useful in bounding the total weight to any subset after an arbitrary number of merges have been performed. We prove this in Lemma 4.

[3] d(u, A) is calculated for all $v \in A$ and not $v \in A \backslash \{u\}$. For $A = V$, we use the notation $d(u)$ to represent $d(u, V)$.

Algorithm 9 Weighted Neighborhood Linkage

Input: similarity function $\mathcal{K}$, set of points S, $\alpha > 0$.

Let the initial threshold $t = 4\alpha n + 1$.

1. Construct a directed complete graph $G = (V, E)$ such that $V = S$. Set $|u| = 1\ \forall u \in V$. Set edge weights $wt(u, v) = 1\ \forall v \in N_t(u)$ and 0 otherwise.
2. Find two nodes u, v such that $argmax_{u,v \in V} C(u, v) \geq t - 2\alpha n$. Break ties arbitrarily.
3. If found u and v, merge u, v using Algorithm 10. *Goto step* 2.
4. If $|V| > 1$, then threshold $t = t + 1$, update graph G as follows and *goto step* 2.
   (a) For each element $x \in S$, let y be the $t + 1_{th}$ nearest neighbor of x in S. Let $x \in u, y \in v;\ u, v \in V$. Update weight $wt(u, v) = wt(u, v) + 1/|u|$.

Output: Tree T with single elements as leaves and internal nodes corresponding to the merges performed.

Algorithm 10 Merge Vertices

Input: vertices u, v, Graph G.

1. Define a new vertex u' such that $u' = u \cup v$.
2. For all vertices $w \in V$ construct edges (u', w) and (w, u') and set weights as follows:
   - $wt(u', w) = \frac{1}{|u'|}(|u|wt(u, w) + |v|wt(v, w))$.
   - $wt(w, u') = wt(w, u) + wt(w, v)$.
3. Update $V = V \backslash \{u, v\} \cup \{u'\}$.

Output: Graph G

Lemma 4. *For a given graph G, Assume $u, v \in V$ both have a total edge weight of f to a subset $A \subseteq V$. If we merge u, v in Algorithm 9 to get a new vertex u', then u' also has a total edge weight of at most f to this subset A.*

Proof: Let us consider two vertices $u, v \in V$ and assume both u, v have a total edge weight of f to a subset $A \subseteq V$, *i.e.* $d(u, A) = d(v, A) = f$.

Now, let us assume we merge u, v to get a new vertex u'. We compute the edge weight from the new vertex u' to any vertex $w \in V$ as $wt(u', w)$ as $|u|wt(u, w) + |v|wt(v, w))/|u'|$.

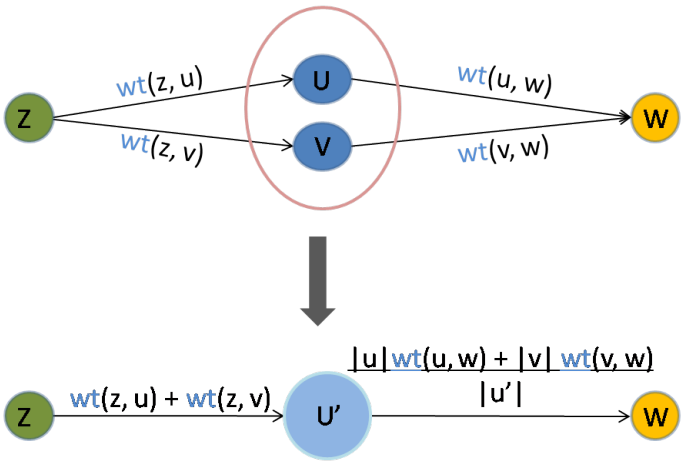


Figure 26: Step 3 - updating graph after merging two vertices

Now, we compute the total edge weight from u' to set $A \subseteq V$.

$$\begin{aligned} d(u') &= \sum_{w \in A} wt(u', w) \\ &= \sum_{w \in A} \frac{|u| wt(u, w) + |v| wt(v, w)}{|u'|} \\ &= \frac{|u|}{|u'|} \sum_{w \in A} wt(u, w) + \frac{|v|}{|u'|} \sum_{w \in A} wt(v, w) \\ &= \frac{|u|}{|u'|} d(u, A) + \frac{|v|}{|u'|} d(v, A) = \frac{|u|}{|u'|} f + \frac{|v|}{|u'|} f \\ &= \frac{|u| + |v|}{|u'|} f = f \end{aligned}$$

■

We now consider the total edge weight that is possible for any vertex at any given time. This is equivalent to it's weighted out-degree. We maintain the invariant that the weighted out-degree from any vertex $u \in V$ is exactly equal to the current value of the threshold. We prove this in Lemma 5.

Lemma 5. *For a given graph G, each vertex has a weighted out-degree exactly equal to the current threshold. i.e.* $\forall u \in V, d(u) = t$.

Proof: We prove this by induction. Note that at $t = 4\alpha n + 1$, step 1 of Algorithm 9 constructs G by adding t edges for each vertex $u \in V$ to it's t nearest neighbors and setting

these edge weights to 1. Thus, we get the weighted out-degree from u as $d(u) = t$. This represents the total edge weight to all the vertices in V.

Now, for a graph G we have for each vertex $u, d(u) = t$. We update the graph either in step 3 or step 4 of the algorithm.

In step 3, we merge two vertices u, v to get a vertex u'. We know that $d(u) = t$ and $d(v) = t$. Then from Lemma 4 we know that $d(u') = t$.

Now, let us update the weights of the incoming edges to u, v. For a vertex $w \in V$ we have

$$\begin{aligned} d(w) &= \sum_{x \in V} wt(w, x) \\ &= wt(w, u') + \sum_{x \in V \setminus \{u'\}} wt(w, x) \\ &= wt(w, u) + wt(w, v) + \sum_{x \in V \setminus \{u'\}} wt(w, x) = t \end{aligned}$$

In step 4, we update graph G. For each for each element $x \in S$, we find y such that y is the $t + 1_{th}$ nearest neighbor of x in S. Let $x \in u, y \in v;\ u, v \in V$. We increase the weight of the edge (u, v) by $1/|u|$. Thus, the total increase in weighted out-degree is $\sum_{x \in u} 1/|u| = 1$. ■

We wish to determine the maximum total edge weight to *bad neighbors* for each vertex $u \in V$. We consider a neighbor $v \in V$ of the vertex $u \in V$ bad, if u has a non-zero edge to v such that u, v contain points belonging to different target clusters. *i.e.* $u \subseteq C_i, v \subseteq C_j, i \neq j$. We bound the total edge weight to bad neighbors in Lemma 6.

Lemma 6. *Let $\mathcal{K}$ be a similarity function satisfying the α-good neighborhood for the clustering problem (S, ℓ). Assume that for each vertex in V that contains points from a cluster C_i, has points only from C_i; i.e. for any $u \in V$ such that $u \cap S \subseteq C_i$, for some $i \in \{1, 2, ...k\}$. Then, for $t \leq n_{C_i}$, $d(u, S \setminus C_i) \leq \alpha n$.*

Proof: We prove this by induction. Assume that there is a vertex $u \in V$ such that $u \cap S \subseteq C_i$ In the beginning, we have $V = S$ and we construct t edges for each vertex $u \in V$, each with weight 1. Since, $\mathcal{K}$ satisfies the α-good neighborhood by Claim 1 u cannot have

edges to more than αn vertices that are not from C_i. Consider the worst case when that at $t = 4\alpha n + 1$, each $u \in V$ has an edge to each of the αn vertices that are not from C_i. Then $d(u, S \setminus C_i) = \sum_{v \in S \setminus C_i} wt(u, v) = \alpha n$. Now, if we merge u, v to get a new vertex u', we know from Lemma 4 that $d(u', S \setminus C_i) = \alpha n$.

At any $t \leq n_{C_i}$, by Claim 1, all the new neighbors that get added in graph G will belong to C_i. Thus, $d_t(u, S \setminus C_i) = d_{t+1}(u, S \setminus C_i) = \alpha n$. Thus, in the worst case, we never have $d(u, S \setminus C_i) > \alpha n$. ■

The lemma gives some useful information about the total edge weight to *good neighbors* of each vertex. We consider a neighbor $v \in V$ of the vertex $u \in V$ good, if u, v contain points belonging to same target clusters. *i.e.* $u \subseteq C_i, v \subseteq C_j, i = j$. Corollary 1 gives the total edge weight to good neighbors.

Corollary 1. *Let $\mathcal{K}$ be a similarity function satisfying the α-good neighborhood for the clustering problem (S, ℓ). Assume that for each vertex in V that contains points from a cluster C_i, has points only from C_i; i.e. for any $u \in V$, we have $u \cap S \subseteq C_i$, for some $i \in \{1, 2, ...k\}$. Then, for $t \leq n_{C_i}$, $d(u, C_i) \geq t - \alpha n$.*

Proof: The proof follows directly from Lemmas 5 and 6 ■

So far, we have been able to bound the total edge weight to *bad neighbors* for a vertex when t is less than the size of the target cluster to which it belongs. Lemma 7 gives a useful way of bounding the total edge weight to *bad neighbors* after $t > n_{C_i}$.

Lemma 7. *Let $\mathcal{K}$ be a similarity function satisfying the α-good neighborhood for the clustering problem (S, ℓ). Assume that for each vertex in V that contains points from a cluster C_i, has points only from C_i; i.e. for any $u \in V$, we have $u \cap S \subseteq C_i$, for some $i \in \{1, 2, ...k\}$. Then, for $t \geq n_{C_i}$, $d(u, S \setminus C_i) \leq \alpha n + t - n_{C_i}$.*

Proof: From Lemma 6, we know that for $t \leq n_{C_i}$, $d(u, S \setminus C_i) \leq \alpha n$.

In step 4, when we update graph G, for each for each element $x \in S$, we find y such that y is the $t + 1_{th}$ nearest neighbor of x in S. For $t > n_{C_i}$, we know that $y \in S \setminus C_i$.

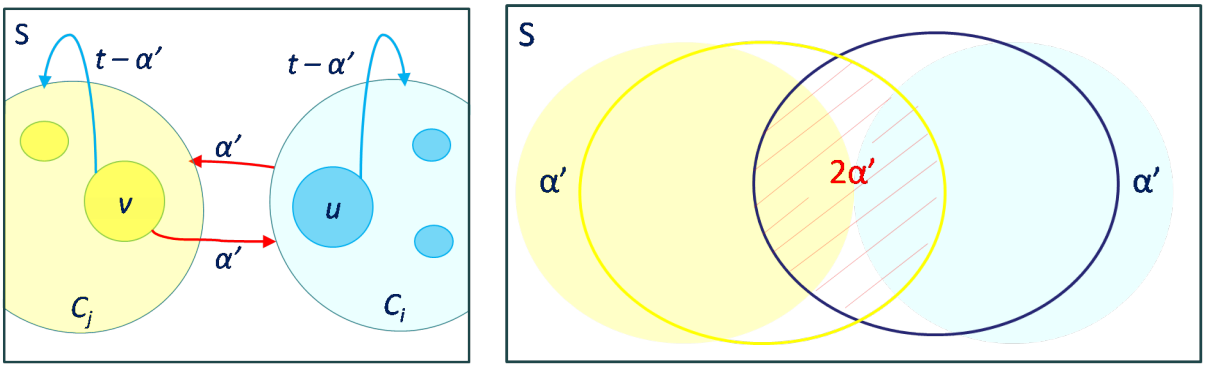
For $x \in u, y \in v$; $u, v \in V$, we increase the weight of the edge (u, v) by $1/|u|$. Thus, the total increase in weights over all points in u is $\sum_{x \in u} 1/|u| = 1$. Thus, $d_{t+1}(u, S \setminus C_i) = d_t(u, S \setminus C_i) + 1 \leq \alpha n + t - n_{C_i}$. ■

Now that we know the total edge weights under different conditions, we use this information to maintain two invariants while merging:

1. never merge two vertices belonging to different target clusters (Figures 27 and 28).

2. by $t = n_{C_i}$ merge all vertices belonging to the target cluster C_i (Figure 29).

First, we show that we never merge two vertices belonging to different target clusters till the t is large enough such that we would have had sufficient time to merge all the vertices belonging to the same target cluster (Figures 27 and 28). This is proved in Lemmas 8 and 9.



(a) $d(.,.)$ (b) $C(u, v)$

Figure 27: Maximum $C(u, v)$ between two nodes u, v belonging to different target clusters, *i.e.* $u \subseteq C_i, v \subseteq C_j$ cannot be more than $2\alpha n$. ($\alpha' = \alpha n$)

Lemma 8. *Let $\mathcal{K}$ be a similarity function satisfying the α-good neighborhood for the clustering problem (S, ℓ). Then, in Algorithm 9, we never merge two vertices belonging to different clusters till t is greater than both the cluster sizes, i.e. for $u, v \in V$,*

$u \cap S \subseteq C_i, v \cap S \subseteq C_j, i \neq j$. As long as long as $t \leq n_{C_i} + n_{C_j} - 3\alpha n$ and the smallest cluster size is greater than $4\alpha n$, we never merge u, v.

Proof: We can only merge two vertices if $C(u, v) \geq t - 2\alpha n$ where

$$C(u, v) = \sum_{w \in V} min(wt(u, w), wt(v, w))$$

. Assume without loss of generality $n_{C_i} < n_{C_j}$.

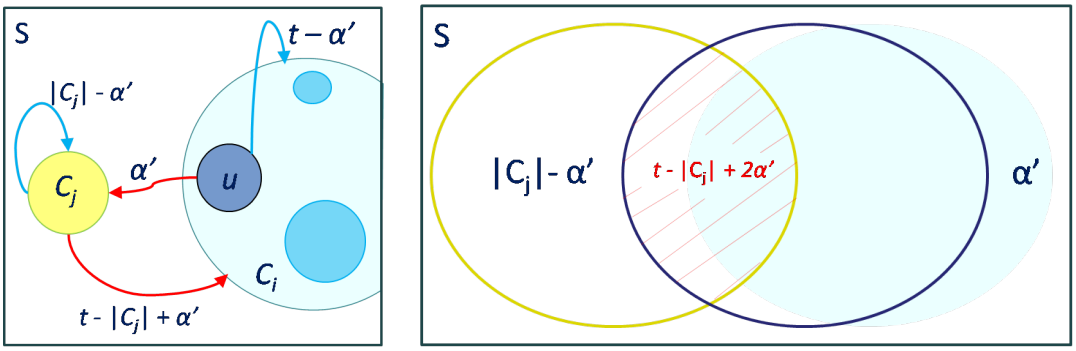
From Lemma 6, for $t \leq n_{C_i}$, we know that $d(u, S \setminus C_i) \leq \alpha n$ and $d(v, S \setminus C_j) \leq \alpha n$. Thus, $C(u, v) \leq 2\alpha n$. Since, $t > 4\alpha n + 1$, $t - 2\alpha n > C(u, v)$. Thus, we can never merge two vertices from different clusters.

From Lemma 7, $n_{C_i} < t \leq n_{C_j}$, we know that $d(u, S \setminus C_i) \leq \alpha n + t - n_{C_i}$. Thus, $C(u, v) \leq 2\alpha n + t - n_{C_i}$. Since, $t > 4\alpha n + 1$, for $n_{C_i} > 4\alpha n$, $t - 2\alpha n > C(u, v)$.

For $n_{C_j} < t$, we know that $d(v, S \setminus C_i) \leq \alpha n + t - n_{C_j}$.

For $t \leq n_{C_i} + n_{C_j} - 3\alpha n$ Thus, $C(u, v) \leq d(u, S \setminus C_i) + d(v, S \setminus C_j) < t - 2\alpha n$.

Thus, we never merge u and v as long as $t < n_{C_i} + n_{C_j} - 3\alpha n$, which is greater than both n_{C_i} and n_{C_j}. ■



(a) $d(.,.)$ (b) $C(u, C_j)$

Figure 28: Maximum $C(u, C_j)$ between a node $u \subseteq C_i$ and a fully formed target cluster C_j cannot be more than $t - |C_j| + 2\alpha n$. ($\alpha' = \alpha n$)

Lemma 9. *Let $\mathcal{K}$ be a similarity function satisfying the α-good neighborhood for the clustering problem (S, ℓ). Then, in Algorithm 9, we never merge two vertices that do not belong*

to the same cluster. i.e. for $u, v \in V$, $u \cap S \subseteq C_i, v \cap S = C'$ such that $C' = \bigcup_{j \in L} C_j$ where $L \subseteq [k] \backslash \{i\}$. As long as long as $t \leq n_{C_i}$ and the smallest cluster size is greater than $4\alpha n$, we never merge u, v.

Proof: To merge two vertices, we require the condition $C(u, v) \geq t - 2\alpha n$ where $C(u, v) = \sum_{w \in V} min(wt(u, w), wt(v, w))$.

From Lemma 6, for $t \leq n_{C_i}$, we know that $d(u, S \backslash C_i) \leq \alpha n$.

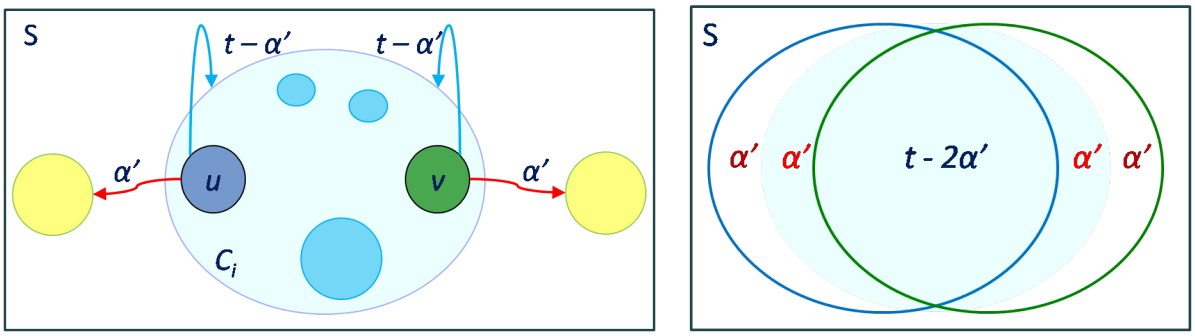
Let us assume the vertex v is formed as a result of a union of several target clusters that do not contain any points from the cluster C_i, *i.e.* $v \cap S = C'$ such that $C' = \bigcup_{j \in L} C_j$ where $L \subseteq [k] \backslash \{i\}$. From Claim 1 we know that each point in each contributing cluster C_j could not have more than αn neighbors outside C_j. Since $C_j \subseteq C'$, we know that each point could have at most αn neighbors outside C'. From Lemma 4 we know that after combining these contributing clusters $d(v, C') \leq \alpha n$. Combining with Lemmas 6 and 7 we get $d(v, S \backslash C') \leq \alpha n + t - n_{C'}$.

Thus, $C(u, v) \leq 2\alpha n + t - n_{C'}$. Since, $t > 4\alpha n + 1$, $t - 2\alpha n > C(u, v)$ for $n_{C'} > 4\alpha n$. Since we know that the size of the smallest cluster is greater than $4\alpha n$, $n_{C'}$ must also be greater than $4\alpha n$.

Thus, we never merge u and v for $t \leq n_{C_i}$. ■

Now, we show that it is possible to merge any two vertices belonging to the same target cluster for a threshold that is sufficiently smaller than the threshold for merging two vertices belonging to different target clusters. Particularly, we show that this threshold is at most the size of the target cluster to which the vertex belongs (Figure 29). Lemma 10 gives the proof.

Lemma 10. *Let $\mathcal{K}$ be a similarity function satisfying the α-good neighborhood for the clustering problem (S, ℓ). In Algorithm 9, at iteration $t = n_{C_i}$, we can merge two vertices belonging to the cluster C_i, i.e. $u, v \in V$, such that $u \cap S \subseteq C_i, v \cap S \subseteq C_i$, we can merge u, v.*



(a) $d(.,.)$ (b) $C(u,v)$

Figure 29: Minimum $C(u,v)$ between two nodes u, v belonging to same target clusters, *i.e.* $u, v \subseteq C_i$ must be at least $t - 2\alpha n$. ($\alpha' = \alpha n$)

Proof: Let us consider the set of vertices $U_i \subseteq V$ such that it contains all vertices w that belong to the cluster C_i, *i.e.* $w \cap S \subseteq C_i$.

Let us look at the case when $t = n_{C_i}$. For a vertex $w \in U_i$, we look at the weight of the edge $e = (u, w)$. Let $P_t(x, v)$ be the set of points out of the t nearest neighbors of a point $x \in S$ that lie in vertex $v \in V$, *i.e.* $P_t(x, v) = v \cap N_t(x)$. A point $x \in u$ will have $P_t(x, w)$ points in the vertex w. It is easy to note that $max(0, |w| - \alpha n) \leq |P_t(x, w)| \leq |w|$. This follows directly from Claim 1 that each point $x \in C_i$ can have at most αn points outside C_i. The total weight the point x contributes to $wt(e)$ is $|P_t(x, w)|/|u|$. We consider the set of points $B_t(x, w)$ such that $B_t(x, w) = w \backslash P_t(x, w)$. Then we can re-write the contribution of x as $(|w| - |B_t(x, w)|)/|u|$ Thus, $wt(e) = \sum_{x \in u}(|w| - |B_t(x, w)|)/|u|$. Similarly for vertex v, the weight of the edge $e' = (v, w)$ is $wt(e') = \sum_{y \in v}(|w| - |B_t(y, w)|)/|v|$.

Now we look at the difference in weights for the two edges e and e'. We divide the set U_i into two subsets $U_i', U_i'' \subseteq U_i$ such that for all vertices $w' \in U_i'$, we have $wt(u, w') \geq wt(v, w')$ and for all vertices $w'' \in U_i''$, we have $wt(u, w'') < wt(v, w'')$.

Let us compute the difference $\Delta(u, v)$ in weights over all vertices in U_i''. Note that

$\sum_{w'' \in U_i''} |w''| = |U_i''|$.

$$\begin{aligned}
\Delta(u,v) &= \sum_{w'' \in U_i''} (wt(v,w'') - wt(u,w'')) \\
&= \sum_{w'' \in U_i''} \left(\sum_{y \in v} \frac{|w''| - |B_t(y,w'')|}{|v|} - \sum_{x \in u} \frac{|w''| - |B_t(x,w'')|}{|u|} \right) \\
&= \sum_{y \in v} \sum_{w'' \in U_i''} \frac{|w''| - |B_t(y,w'')|}{|v|} - \sum_{x \in u} \sum_{w'' \in U_i''} \frac{|w''| - |B_t(x,w'')|}{|u|} \\
&= \sum_{y \in v} \frac{|U_i''|}{|v|} - \sum_{y \in v} \sum_{w'' \in U_i''} \frac{|B_t(y,w'')|}{|v|} - \sum_{x \in u} \frac{|U_i''|}{|u|} + \sum_{x \in u} \sum_{w'' \in U_i''} \frac{|B_t(x,w'')|}{|u|} \\
&= |U_i''| - \sum_{y \in v} \frac{\sum_{w'' \in U_i''} |B_t(y,w'')|}{|v|} - |U_i''| + \sum_{x \in u} \frac{\sum_{w'' \in U_i''} |B_t(x,w'')|}{|u|} \\
&= \sum_{x \in u} \frac{\sum_{w'' \in U_i''} |B_t(x,w'')|}{|u|} - \sum_{y \in v} \frac{\sum_{w'' \in U_i''} |B_t(y,w'')|}{|v|}
\end{aligned}$$

$\Delta(u,v)$ is maximized when $\sum_{w'' \in U_i''} |B_t(x,w'')|$ is maximum and $\sum_{w'' \in U_i''} |B_t(y,w'')|$ is minimum. But, $\sum_{w'' \in U_i''} |B_t(x,w'')| \leq \sum_{w \in U_i} |B_t(x,w)|$ and from Claim 1 we know that $\sum_{w \in U_i} |B_t(x,w)| \leq \alpha n$. Thus, we get

$$\begin{aligned}
\Delta(u,v) &= \sum_{x \in u} \frac{\sum_{w'' \in U_i''} |B_t(x,w'')|}{|u|} - \sum_{y \in v} \frac{\sum_{w'' \in U_i''} |B_t(y,w'')|}{|v|} \\
&\leq \sum_{x \in u} \frac{\alpha n}{|u|} - \sum_{y \in v} \frac{0}{|v|} \\
&= \alpha n \sum_{x \in u} \frac{1}{|u|} = \alpha n
\end{aligned}$$

Now, we compute $C(u,v)$ for the two vertices.

$$\begin{aligned}
C(u,v) &= \sum_{w \in V} min(wt(u,w), wt(v,w)) \\
&= \sum_{w \in U_t} min(wt(u,w), wt(v,w)) + \sum_{w \in V \setminus U_t} min(wt(u,w), wt(v,w)) \\
&\geq \sum_{w \in U_t} min(wt(u,w), wt(v,w)) \\
&= \sum_{w' \in U_t'} wt(v,w') + \sum_{w'' \in U_t''} wt(u,w')
\end{aligned}$$

$$= \sum_{w' \in U'_t} wt(v, w') + \sum_{w'' \in U''_t} (wt(v, w')) - \Delta(u, v)$$
$$= \sum_{w \in U_t} wt(v, w) - \Delta(u, v)$$

From Corollary 1 we know that $d(u, C_i) \geq t - \alpha n$.

$$C(u, v) \geq t - \alpha n - \alpha n = t - 2\alpha n$$

Thus, the merging condition is satisfied for u, v. ■

From Lemma 10 we know that we can merge definitely merge any two vertices belonging to the same target cluster. Thus, we should be able to merge all such vertices to form a single vertex that represents the entire target cluster. We formalize this in the Theorem 9.

Theorem 9. *Let $\mathcal{K}$ be a similarity function satisfying the α-good neighborhood for the clustering problem (S, ℓ). As long as the smallest cluster size is greater than $4\alpha n$, then, in Algorithm 9, by iteration $t = n_{C_i}$, we would merge all points $x \in C_i$ to form a single vertex that does not contain any points from any other cluster. i.e. $\exists u \in V$, such that $u \cap S = C_i$.*

Proof: We prove this by induction. At each step we show that we satisfy the two invariants:

1. Never merge two vertices belonging to different target clusters.

2. By $t = n_{C_i}$ merge all vertices belonging to the target cluster C_i.

Assume, without loss of generality that $n_{C_1} \leq n_{C_2} \ldots \leq n_{C_k}$.

Let $U_i = \{u_{i_1}, u_{i_2}, \ldots u_{i_m}\}$ denote the vertices such that $u_{i_j} \cap C_i \neq \phi$.

We look at the iteration $t = n_{C_1}$. From Lemma 8, we know that we would not have merged any vertex $x \cap S \subseteq C_1$ with any other vertex y such that $(y \cap S) \cap C_1 = \phi$. Thus, each vertex $u_{1_j} \in U_1$ will contain only points from C_i, *i.e.* $u_{1_j} \cap S \subseteq C_1$. From Lemma 10, we know that we can merge any two vertices $u_{1_j}, u_{1_l} \in U$. This merge gives a new vertex u'. Now it is easy to see that u' can be added to the set U as $u' \cap S = (u_{1_j} \cup u_{1_l}) \cap S = (u_{1_j} \cap S) \cup (u_{1_l} \cap S) \subseteq C_1$. Thus, u' can again be merged with any $u_{1_j} \in U$ until $|U| = 1$.

Therefore, at $t = n_{C_1}$ all vertices belonging to the cluster C_1 will merge to form one vertex u that contains points from only C_1, *i.e.* $u \cap S = C_1$.

Now let us assume that the invariants still hold at $t = n_{C_{i-1}}$. For $n_{C_{i-1}} < t \leq n_{C_i}$, from Lemmas 8 and 9, we know that we would not merge any vertex $x \cap S \subseteq C_i$ with any other vertex y such that $(y \cap S) \cap C_i = \phi$. Thus, each vertex $u_{i_j} \in U$ will contain only points from C_i, *i.e.* $u_{i_j} \cap S \subseteq C_i$. At $t = n_{C_i}$, from Lemma 10, we know that we can merge any two vertices $u_{i_j}, u_{i_l} \in U$. This merge gives a new vertex u'. Now it is easy to see that u' can be added to the set U as $u' \cap S = (u_{i_j} \cup u_{i_l}) \cap S = (u_{i_j} \cap S) \cup (u_{i_l} \cap S) \subseteq C_i$. Thus, u' can again be merged with any $u_{i_j} \in U$ until $|U| = 1$. Therefore, at $t = n_{C_i}$ all vertices belonging to the cluster C_i will merge to form one vertex u that contains points from only C_i, *i.e.* $u \cap S = C_i$. ■

After merging all vertices to form a single vertex u at $t = n_{C_i}$, the first question that comes to mind is that how is u any different from any other vertex and why would this now not start merging with other available clusters. We have already given a formal argument for this in Lemma 8. To further explain, we present Corollary 2.

Corollary 2. *Let $\mathcal{K}$ be a similarity function satisfying the α-good neighborhood for the clustering problem (S, ℓ). Assume that vertex v is formed at $t = n_{C_i}$ such that $v \cap S = C_i$. Then, for $t \geq n_{C_i}$, u has a self edge of weight at least $n_{C_i} - \alpha n$.*

Proof: We know from Theorem 9 that at $t = n_{C_i}$, we merge all vertices u, such that $u \cap S \subseteq C_i$ to form a single vertex v such that $v \cap S = C_i$. From Lemma 5, we know that $d(v) = t$. and from Lemma 7 we know that $d(v, S \setminus C_i) \leq \alpha n + t - n_{C_i}$. Thus, we get $d(v, C_i) \geq n_{C_i} - \alpha n$. Thus, there must exist a self edge of weight at least $n_{C_i} - \alpha n$ for the vertex v. ■

Corollary 2, coupled with the definition of $C(u, v)$ should lay any doubts to rest why the newly formed vertex refrains from merging with vertices belonging to another cluster, thus giving them sufficient time to merge together.

We have shown above that as the threshold reaches a particular value, if there was a

target cluster of size equal to the threshold we would have been able to create a vertex that represented the target cluster. Thus, we have a way of generating all the possible target clusters for the given data set. We formalize this in Theorem 10.

Theorem 10. *Let $\mathcal{K}$ be a similarity function satisfying the α-good neighborhood for the clustering problem (S, ℓ). As long as the smallest cluster size is greater than $4\alpha n$, then, in Algorithm 9, by iteration t, we must have generated each possible target cluster for S satisfying α-good neighborhood of size up to t.*

Proof: The proof follows directly from the Theorem 9. Note that S is also a valid target cluster at $t = |S|$ as the entire data set satisfies the α-good neighborhood we generate this as the last cluster. ■

We keep a track of each merging step in the algorithm and create a hierarchy corresponding to merges. This hierarchy consists of all the possible target clusters that can be formed for the data set. Thus, given any target clustering, we should be able to find it in the hierarchy. We formalize this in Theorem 11.

5.2.1 The Main Result

Theorem 11. *Let $\mathcal{K}$ be a similarity function satisfying the α-good neighborhood for the clustering problem (S, ℓ). As long as the smallest cluster size is greater than $4\alpha n$, then, we can use Algorithm 9 to generate a hierarchy that the ground-truth clustering is a pruning of this tree in $O(n^3)$ time.*

Proof: The proof follows directly from Theorems 10 and 12. ■

5.3 Run Time Analysis

Theorem 12. *Algorithm 9 has a running time of $O(n^3)$.*

Proof: We analyze the different steps of the algorithm separately below.

Preprocessing

In the preprocessing step, we construct a list of nearest neighbors for each point in S by sorting $n-1$ other points in decreasing order of similarity, the point itself being the first member of the list. This list gives $N_t(x)$ which is the first t elements of the list. This takes $O(n \log n)$ time for each point and thus the entire preprocessing step costs $O(n^2 \log n)$ time.

We define two tables. Initially both tables are of size n. The first table VT gives for each point $i \in S$ the index of the vertex $u \in V$ to which it belongs. Initially each point is it's own vertex, thus $VT(i) = i$. The second table SZ gives the size of the vertex $u \in V$. Initially each vertex has size one. Thus $SZ(i) = 1$. We can setup each of these tables in n time.

We define an n x n matrix A to represent the adjacency matrix of the graph G which begins with all the n points as vertices initially. For each of the n vertices, we initialize A by setting $a_{ij} = 1 \; \forall j \in N_t(i)$ and 0 otherwise. This takes a total of n^2 time.

Then we compute an n x n matrix M where m_{ij} gives the weighted common neighbors $C(i, j)$ between vertices i and j. For this we just look at the first t neighbors of either i or j (assume i) and for each neighbor k if $a_{jk} = 1$ increment m_{ij} by one. This requires a total of t time for each pair. Since there are $\binom{n}{2}$ pairs, the total time is $\binom{n}{2}t$. Notice that M is a symmetric matrix and thus we only discuss about updating m_{ij}.

Main Loop

Now we look at the various steps of the algorithm and analyze the time required for each step. Let us assume that at a particular step we have r vertices where $r \leq n$. The matrices A, M would be of size r x r.

Merging Step

First we choose the maximum m_{ij} in M to determine whether we can merge i, j or not. This can be done in r^2 time.

Now let us look at the time taken to update M and A if we decide to merge. Let us assume we are merging two vertices i and j.

First we update the matrix M. We compute a temporary row $AR_{i'}$ such that $AR_{i'} =$

$(SZ(i)row(A_i) + SZ(j)row(A_j)/(SZ(i) + SZ(j))$, and a temporary column $AC_{i'}$ such that $AC_{i'} = cloumn(A_i) + cloumn(A_j)$.

For all vertices k, l that had non-zero edges (k, i) or (k, j) and (l, i) or (l, j) this step will change the value of $C(k, l)$. Earlier the contribution of i and j individually was $min(a_{ki}, a_{li}) + min(a_{kj}, a_{lj})$. Now the contribution after the merge should be $min(a_{ki} + a_{kj}, a_{li} + a_{lj})$. We can do this by collecting all vertices that had non-zero edge weights for i and j and for each pair of vertices k, l, where $k, l \neq j$, we set $m_{kl} = m_{kl} - min(a_{ki}, a_{li}) + min(a_{kj}, a_{lj}) + min(AC_{i'}(k), AC_{i'}(l))$. This takes a total of $\binom{r}{2}$ time.

We also need to update $row(M_i)$. For each vertex k we update m_{ik} as $m_{ik} = m_{ik} - min(row(A_k), row(A_i)) + min(row(A_k), AC_{i'})$. This takes r time per vertex. Since there are r vertices, it takes a total of r^2 time.

Now we can update A. We update row i as $row(A_i) = AR_{i'}$. We update column i as $column(A_i) = AC_{i'}$. Both these steps can be done in r time each. Now we delete $row(A_j)$ and $column(A_j)$ to remove vertex j.

For each point x in j we set $VT(x) = i$. We can do this in $SZ(j)$ time. We can update $SZ(i) = SZ(i) + SZ(j)$ and delete the j_{th} element of SZ . This takes $O(1)$ time.

We do at most $n - 1$ merge steps. Thus the complexity is $O(n^3)$.

Increasing Threshold t

Now we look at the case when we increase the threshold. For each element x in vertex i we find the $(t + 1)_{th}$ nearest neighbor of x. Let's say it is y. We find the vertex to which y belongs to from $VT(y)$. Let's say the vertex is j. We collect all such vertices in the set X. We need to increase the weight of the edge (i, j) by $1/SZ(i)$. We compute the total increase in weight of the edge (i, j), let's say it is $w'(i, j)$.

Now we need to update M with respect to the change. We look at the column j and for each vertex k that has a non-zero entry we update m_{ik} as $m_{ik} = m_{ik} - min(a_{ij}, a_{kj}) + min(a_{ij} + w'(i, j), a_{kj}))$. Now we can increment $a_{ij} = a_{ij} + w'(i, j)$. This can be done in $O(r)$ time. Note that each point in u can find at most 1 such new vertex j. Thus the total number of

such columns that need to be checked is $SZ(i)$. This is done for all vertices. Thus, it takes a total of $O(Nr)$ time to do this step.

We increment the value of t at most $O(n)$ times. Thus, total complexity is $O(n^3)$.

Thus, the total running time for the algorithm is $O(n^3)$. ■

5.4 Discussion

In this chapter we proposed and analyzed a new, more practical, robust algorithm for bottom-up agglomerative clustering that alleviates the limitations of *RHL* by being completely agglomerative in nature, much simpler, faster and more robust to parameter tuning (as we show in Chapter 7). We proved that our algorithm can be used to cluster accurately in cases where the data satisfies the α-good neighborhood property whereas the traditional agglomerative algorithms fail as we have already seen in Chapter 3.

WNL too can be easily extended to an inductive setting with similar correctness guarantees (to be discussed in Chapter 6), where a small subset of points is randomly chosen from a much larger instance space to generate a hierarchy into which the rest of the points are then inserted resulting in a hierarchy over the entire instance space.

We also observe in Chapter 7 that experimentally *WNL* algorithm does much better than other agglomerative algorithms and comparably to *RHL* even though we only use a single parameter α to characterize noise. Not only is *WNL* provably faster than *RHL*, it can also be shown experimentally to be more robust to incorrect estimates of input parameter α as compared to *RHL*.

CHAPTER VI

THE INDUCTIVE SETTING

For large data sets having a large number of observations, it is often resource and time intensive to run an algorithm over the entire data set, even impossible if enough resources are not available. In such cases, it is necessary to have an algorithm that can be run inductively over the data set. In the inductive setting, a small subset of points is randomly chosen from a much larger instance space to generate a hierarchy into which the rest of the points are then inserted resulting in a hierarchy over the entire instance space.

There are no known ways of extending the standard linkage algorithms to the inductive setting and thus have to be run over the entire data set. On the other hand, both our algorithms, *RHL* and *WNL* can be easily extended to an inductive setting. Moreover, they require only a small random sample which is independent on the size of the instance space and depends only on the noise parameters α, ν and the confidence parameter δ.

In the following sections we describe the inductive versions of both *RHL* and *WNL* algorithms and give formal theoretical guarantees for these inductive algorithms. First we formally define the framework and goals in the inductive setting.

6.1 Formal Definition

We consider an inductive model in which S is merely a small random subset of points from a much larger abstract instance space X. The goal here is to generate a hierarchy over the sample S, which also implicitly represents a hierarchy of the whole space with respect to the underlying distribution, and then insert into it the rest of the points in X to generate a hierarchy over the entire instance space.

For simplicity, we assume that X is finite and that the underlying distribution is uniform over X.

Notation:

In addition to the usual notation, we define several new symbols for the inductive setting.

Let $N = |X|$, *i.e.* the size of the entire instance space.

Let function $f : X \to \{0, 1\}$ denote the induced clusters. Let T represent a hierarchy. For each $x \in X$ and $u \in T$, $f_u(x) = 1$ if x is a point in u and 0 otherwise.

Let $NN(x)$ be the nearest $n_{C(x)}$ nearest neighbors of x in X and $\tilde{N}_t(x)$ denote the t nearest neighbors of x in S.

6.2 Robust Hierarchical Linkage

In this section we describe the inductive version of the *Robust Hierarchical Linkage* algorithm. The procedure is described in Algorithm 11.

Algorithm 11 Inductive Robust Hierarchical Linkage

Input: Similarity function $\mathcal{K}$, parameters $\alpha, \nu, \epsilon > 0$, $k \in Z^+$; $n = n(\epsilon, \gamma, k, \delta)$;

- Pick a set $S = \{x_1, \ldots, x_n\}$ of n random examples from X.
- Run Algorithm 8 with parameters 2α, 2ν on the set S and obtain a tree T on the subsets of S. Let Q be the set of leaves of this tree.
- Associate each node u in T a function f_u (which induces a cluster) specified as follows:

  Consider $x \in X$, and let $q(x) \in Q$ be the leaf given by $\text{argmax}_{q \in Q} \mathcal{K}_{\text{median}}(x, q)$; if u appears on the path from $q(x)$ to the root, then set $f_u(x) = 1$, otherwise set $f_u(x) = 0$.
- Output the tree T.

Our main result in this section is the following:

Theorem 13. *Let $\mathcal{K}$ be a symmetric similarity function satisfying the (α, ν)-good neighborhood for the clustering problem (X, ℓ). As long as the smallest target cluster has size greater than $18(\nu + \alpha)N$, then using Algorithm 11 with parameters $n = \Theta\left(\frac{1}{\min(\alpha,\nu)} \ln \frac{k}{\delta \cdot \min(\alpha,\nu)}\right)$, we*

can produce a tree with the property that the ground-truth is $2\nu + \delta$*-close to a pruning of this tree with probability* $1 - \delta$*. Moreover, the size of this tree is* $O\left(\frac{1}{\alpha} \ln \frac{n}{\delta}\right)$.

Proof: Note that n is large enough so that by Lemma 11, with probability at least $1-\delta/2$, we have that $\mathcal{K}$ satisfies the $(2\alpha, 2\nu)$-good neighborhood with respect to the clustering induced over the sample, and moreover by Chernoff bounds each target cluster has at least $9(\nu+\alpha)n$ points in the sample. So, by Theorem 7 we get that the tree induced over the sample has error at most 2ν. By Algorithm 6, we also obtain a list L of blobs each of size at least $6(\nu + \alpha)n$ that form a partition of S and each blob in the list L contains good points from only one good set *i.e.* for any $C \in L$, $C \cap G \subseteq G_i$ for some $i \leq k$. Moreover, each blob contains more good points than bad points.

Since we have the right proportion of good and bad points at the leaves of the tree induced over the sample, we get that each new good point with high probability connects to a leaf that contain good points from it's own cluster only. This together with Theorem 7 implies the desired result. ■

Lemma 11. *Let* $\mathcal{K}$ *be a symmetric similarity function satisfying the* (α, ν)*-good neighborhood for the clustering problem* (X, ℓ)*. If* $n = \Theta\left(\frac{1}{\min(\alpha,\nu)} \ln \frac{kn}{\delta}\right)$*, we have* $\mathcal{K}$ *satisfies the* $(2\alpha, 2\nu)$*-good neighborhood with respect to the clustering induced over the sample.*

Proof: First, by Chernoff bounds we immediately have that with high probability, at most $2\nu n$ bad points fall into the sample. Assume that this indeed is the case and we now focus on the α parameter. In fact, for convenience of notation, we assume below that $\nu = 0$. Recovering the general case is immediate.

By assumption we have that $|NN(x) \setminus C(x)| \leq \alpha N$.

We will do a union bound over each point x in the sample. Fix some point $x \in S$ and consider drawing $n = \Theta\left(\frac{1}{\alpha} \ln \frac{n}{\delta}\right)$, random points from X. We are given that:

$$Pr_{z \sim X}[z \in NN(x) \setminus C(x)] \leq \alpha.$$

Since $NN(x)$ and $C(x)$ have the same size, this is equivalent to the statement:

$$Pr_{z\sim X}[z \in C(x) \setminus NN(x)] \leq \alpha.$$

So, by Chernoff bounds applied to both of the above, with high probability at most $2\alpha n$ points in our sample are in $NN(x) \setminus C(x)$ and at most $2\alpha n$ points in our sample are in $C(x) \setminus NN(x)$.

We now argue that at most $2\alpha n$ of the $n_{\tilde{C}(x)}$ nearest points to x in the sample can be outside $C(x)$, where $n_{\tilde{C}(x)}$ is the number of points in the *sample* that are in $C(x)$. Let n_1 be the number points in sample in $NN(x) \setminus C(x)$.

Let n_2 be points in sample in $C(x) \setminus NN(x)$. Let n_3 be points in sample in $C(x) \cap NN(x)$. So, $n_{\tilde{C}(x)} = n_2 + n_3$, and we are given that $n_1, n_2 \leq 2\alpha n$.

There are two cases:

***CASE 1*:** $n_1 \geq n_2$.

So, $n_1 + n_3 \geq n_2 + n_3 = n_{\tilde{C}(x)}$. This implies that the nearest $n_{\tilde{C}(x)}$ points to x in the sample all lie inside $NN(x)$, since by definition all points inside $NN(x)$ are closer to x than any point outside $NN(x)$. But we are given that at most $n_1 \leq 2\alpha n$ of them can be outside $C(x)$. So we are done.

***CASE 2*:** $n_1 \leq n_2$.

This implies that the nearest $n_{\tilde{C}(x)}$ points to x in the sample include *all* the points in $NN(x)$ in the sample, plus possibly some others too. But this implies in particular that it includes all the n_3 points in $C(x) \cap NN(x)$ in the sample. So, it can include at most $n_{\tilde{C}(x)} - n_3 \leq 2\alpha \cdot n$ points not in $C(x) \cap NN(x)$, and even if all those are not in $C(x)$, it is still $\leq 2\alpha n$.

■

6.3 Weighted Neighborhood Linkage

In this section we describe the inductive version of the *Weighted Neighborhood Linkage* algorithm. The procedure is described in Algorithm 12.

Algorithm 12 Inductive Weighted Neighborhood Linkage

Input: Similarity function $\mathcal{K}$, parameters $\alpha, k \in Z^+$; $n = O(\alpha, k, \delta)$;

- Pick a set $S = \{x_1, \ldots, x_n\}$ of n random examples from X.
- Run Algorithm 9 with parameters 2α on the set S and obtain a tree T on the subsets of S.
- Associate each node u in T a function f_u (which induces a cluster) specified as follows:
- For each $x \in X$
  1. Set $u = root(T)$, traverse T top-down:
  2. Set $f_u(x) = 1$.
  3. Let $l = left(u)$ and $r = right(u)$.
  4. Set $u = l$ and goto step 2 if
     i. $|l| > 4\alpha n$ && $\left|l \cap \tilde{N}_{4\alpha n+1}(x)\right| > 2\alpha n$
     ii. $|l| \leq 4\alpha n$ && $|r| \leq 4\alpha n$ && $C_{4\alpha n+1}(x, l) > C_{4\alpha n+1}(x, r)$
  5. Otherwise set $u = r$ and goto step 2.
- Output f

Lemma 12. *Let $\mathcal{K}$ be a similarity function satisfying the α-good neighborhood for the clustering problem (X, ℓ). If we draw S of size $\Theta\left(\frac{1}{\alpha} \ln \frac{1}{\delta\alpha}\right)$ from X, the with probability at least $1 - \delta$, the similarity function K satisfies the 2α-good neighborhood property with respect to the clustering induced over the sample S.*

Proof: For any point $x \in X$, let $NN(x)$ be it's nearest $n_{C(x)}$ neighbors in X. From Claim 1 we have that $|NN(x) \setminus C(x)| \leq \alpha N$.

Let us assume that $n = \Theta\left(\frac{1}{\alpha} \ln \frac{n}{\delta}\right)$. For a given $x \in S$, We know that:

$$Pr_{z \sim X}[z \in NN(x) \setminus C(x)] \leq \alpha$$

Since $NN(x)$ and $C(x)$ have the same size, this is equivalent to the statement:

$$Pr_{z \sim X}[z \in C(x) \setminus NN(x)] \leq \alpha.$$

So, by Chernoff bounds applied to both of the above, with probability at least $1 - \delta/n$ at most $2\alpha n$ points in our sample are in $NN(x) \setminus C(x)$ and at most $2\alpha n$ points in our sample are in $C(x) \setminus NN(x)$.

Let $\tilde{C} = C \cap S$. Let n_1 be the number of points in S in $NN(x) \setminus C(x)$). Let n_2 be the number of points in S in $C(x) \setminus NN(x)$. Let n_3 be the number of points in S in $C(x) \cap NN(x)$. Thus, $n_{\tilde{C}(x)} = n_2 + n_3$ and we know that $n_1, n_2 \leq 2\alpha n$.

We consider two cases.

***CASE 1*:** $n_1 \geq n_2$.

So, $n_1 + n_3 \geq n_2 + n_3 = n_{\tilde{C}(x)}$. This implies that the nearest $n_{\tilde{C}(x)}$ neighbors of x in the sample all lie inside $NN(x)$, since by definition all points inside $NN(x)$ are closer to x than any point outside $NN(x)$. But we are given that at most $n_1 \leq 2\alpha n$ of them can be outside $C(x)$. Thus, we get that at most $2\alpha n$ of the $n_{\tilde{C}(x)}$ nearest neighbors of x are not from $C(x)$.

***CASE 2*:** $n_1 < n_2$.

This implies that the nearest $n_{\tilde{C}(x)}$ neighbors of x in the sample include *all* the points in $NN(x)$ in the sample, and possibly some others too. But this implies in particular that it includes all the n_3 points in $C(x) \cap NN(x)$ in the sample. So, it can include at most $n_{\tilde{C}(x)} - n_3 \leq 2\alpha n$ points not in $C(x) \cap NN(x)$, and even if all those are not in $C(x)$, it is still $\leq 2\alpha n$. Thus, we get that at most $2\alpha n$ of the $n_{\tilde{C}(x)}$ nearest neighbors of x are not from $C(x)$.

Thus, we get that with probability at least $1 - \delta/n$, at most $2\alpha n$ of the $n_{\tilde{C}(x)}$ nearest neighbors of x in S lie outside $C(x)$. Taking a union bound over all points in S, with probability at least $1 - \delta$ all points in S have at most $2\alpha n$ of the $n_{\tilde{C}(x)}$ nearest neighbors of x in S outside $C(x)$.

Using the inequality $\ln x \leq \alpha x - \ln \alpha - 1$ for $\alpha, x > 0$, gives the required bound for n. ■

Theorem 14. *Let $\mathcal{K}$ be a similarity function satisfying the α-good neighborhood for the clustering problem (X, ℓ). As long as the smallest target cluster has size greater than $16\alpha N$, then using Algorithm 12 with parameters $n = \Theta\left(\frac{1}{\alpha} \ln \frac{1}{\delta\alpha}\right)$, we can produce a tree such that the ground-truth is a pruning of this tree with probability at least $1 - \delta$.*

Proof: From Lemma 12 we know that, with probability at least $1 - \delta$, $\mathcal{K}$ satisfies 2α-good neighborhood with respect to the clustering induced over the sample S.

Moreover by Chernoff bounds each target cluster has at least $8(\nu + \alpha)n$ points in S. By Theorem 11, we can use Algorithm 9 to get a hierarchy for the sample S which has no error. This happens with probability $1 - \delta$.

In Algorithm 12, we add a point to a cluster only if it contains at least $2\alpha n + 1$ neighbors out of $4\alpha n + 1$ nearest neighbors in n. We never choose a node $u \in T$ that contains at most $2\alpha n$ neighbors of x and assign it to a node that has enough neighbors so that it represents cluster $C(x)$. Thus, we always assign x to the correct node in each iteration. We do this until size of at least one of the siblings is greater than $4\alpha n$. When the size of both siblings is less than $4\alpha n$ implies that size of the parent is less than $8\alpha n$. Since the size of the smallest cluster is $16\alpha n$, the parent must already be a subset of $C(x)$. Thus, we are already inside the target cluster of x and we cannot make a mistake however we choose between siblings. We simply prefer one with higher number of common neighbors for completeness.

Thus, with probability $1 - \delta$, we assign all points to their right target clusters. ■

6.4 Discussion

In this chapter, we proved that both our algorithms, *RHL* and *WNL* can be easily extended to an inductive setting with formal robustness guarantees. Moreover, Theorems 14 and 13 prove that the size of the random sample required by both the algorithms is independent on the size of instance space and depends only on the noise parameters α, ν and the confidence parameter δ.

It is interesting to note that in both inductive extensions the algorithms for inserting rest of the points into the hierarchy generated on the random sample is much simpler than both *RHL* and *WNL* run on the random sample. Thus, not only does having an inductive extension of an algorithm reduce the space complexity of the problem but also helps in significantly reducing the amortized running time as well. Thus, if we are willing to lose a bit in the accuracy, the analysis in this chapter allows us to speed up the algorithms significantly.

CHAPTER VII

EXPERIMENTS

In this chapter we do a systematic experimental analysis of several hierarchical clustering algorithms evaluate their robustness by comparing their performance by comparing their results on a wide variety of *synthetic* and *real-world* data sets(Frank & Asuncion, 2010) and show that our algorithms based on the good neighborhood property, *i.e.* *RHL* and *WNL*, consistently perform much better and are much more robust to various forms of noise as compared to other hierarchical algorithms.

In Chapter 6, we discussed the inductive extensions of both the *RHL* and *WNL* algorithms. We experimentally validate the efficacy of these extensions by evaluating their results of large data sets where running hierarchical algorithms in a transductive setting (*i.e.* over the entire data set) is computationally prohibitive, both with respect to time and space. This analysis is especially useful as standard hierarchical algorithm cannot even be run on these data sets on simple machines since these algorithms do not have any inductive version.

Both *RHL* and *WNL* algorithms require additional parameters as input. However, as discussed in Chapter 5, estimating these parameters for new data sets, where the size of smallest cluster is unknown, may be difficult. Thus, it is important to discuss the robustness of these algorithms to parameter tuning. We do an analysis of the performance with varying values of the input parameters. We show that though *RHL* is not as robust to parameter tuning and performs well on small range of varying values, *WNL* is more robust to parameter tuning as it much less sensitive to incorrect estimates of α as compared to *RHL*.

7.1 Data Sets

For the experiments, several synthetic and real-world data sets such as *Digits, Breast Cancer Wisconsin (Diagnostic), Spambase, Ionosphere, Mushroom, Iris* and *Wine* were chosen to encompass a wide variety of fields where agglomerative clustering techniques are used extensively. Most of these data sets are available at the *UCI Machine Learning Repository*(Frank & Asuncion, 2010) along with a detailed description of the attributes and properties of each data set.

We briefly describe these data sets below.

7.1.1 Real-World Data Sets

We describe below the real-world sets used in our experiments.

7.1.1.1 Iris

This is perhaps the best known database to be found in the machine learning and pattern recognition literature. (Fisher et al., 1936) paper is a classic in the field and is referenced frequently to this day (*e.g.* (Duda et al., 2000)). The details of the data-set are as follows:

Table 1: Data Set: Iris

Area	Biology	**Data Set Characteristics**	Multivariate
Data Size	150	**Number of Classes**	3
Number of Attributes	4	**Missing Attribute Values**	None
Attribute Characteristics		Continuous, Numeric (Real)	
Number of Instances per Class		50, 50, 50	

This is a small data set with equal size clusters. One class is linearly separable from the other two; the latter are *NOT* linearly separable from each other.

7.1.1.2 Wine

This data set is the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivators. The analysis determined the quantities of 13

constituents found in each of the three types of wines. The details of the data-set are as follows:

Table 2: Data Set: Wine

Area	Physical	**Data Set Characteristics**	Multivariate
Data Size	178	**Number of Classes**	3
Number of Attributes	13	**Missing Attribute Values**	None
Attribute Characteristics		Continuous, Numeric (Integer, Real)	
Number of Instances per Class		59, 71, 48	

This is another small data set with almost similar size clusters.

7.1.1.3 Digits

The $MNIST$ database of handwritten digits has a training set of $60,000$ examples, and a test set of $10,000$ examples. It is a subset of a larger set available from $NIST$. This data set is a well known benchmark for classification and pattern recognition algorithms and has been used extensively in testing clustering algorithms as well (Dasgupta & Long, 2005).

The original black and white (bilevel) images from $NIST$ were size normalized to fit in a 20×20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28×28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28×28 field.

The training set contained examples from approximately 250 writers and the sets of writers of the training set and test set are disjoint. In the test data set, the first 5000 are cleaner and easier than the last 5000.

For our experiments we use the test data set and do not require the training file. The details of the test data-set are as follows:

This data set is a large data set with a lot of noise when clustering all the digits. For simplicity, different subset of numbers were randomly picked as classes and experiments were conducted on the reduced data sets containing images of these numbers only. *e.g.* digit

Table 3: Data Set: Digits

Area	Image Processing	**Data Set Characteristics**	Multivariate
Data Size	10000	**Number of Classes**	10
Number of Attributes	28×28	**Missing Attribute Values**	None
Attribute Characteristics		Discrete $[0, 255]$, Numeric (Integer)	

pairs such as 0&9, 4&7 or larger subsets such as $0, 1, 2, 3, 4$&5 we used as classes.

7.1.1.4 Breast Cancer Wisconsin & Breast Cancer Wisconsin (Diagnostic)

This breast cancer databases was obtained from the *University of Wisconsin Hospitals, Madison*.

In this data set, features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. Separating plane was obtained using Multisurface Method-Tree ($MSM-T$) (Bennett, 1992), a classification method which uses linear programming to construct a decision tree. Relevant features were selected using an exhaustive search in the space of $1-4$ and $1-3$ separating planes.

The details of the Breast Cancer Wisconsin data set are as follows:

Table 4: Data Set: Breast Cancer Wisconsin

Area	Medical Science	**Data Set Characteristics**	Multivariate
Data Size	699	**Number of Classes**	2
Number of Attributes	10	**Missing Attribute Values**	Yes (16)
Attribute Characteristics		Continuous, Numeric (Real)	
Number of Instances per Class		458, 241	

The details of the Breast Cancer Wisconsin (Diagnostic) data-set are as follows:

Table 5: Data Set: Breast Cancer Wisconsin (Diagnostic)

Area	Medical Science	**Data Set Characteristics**	Multivariate
Data Size	569	**Number of Classes**	2
Number of Attributes	32	**Missing Attribute Values**	None
Attribute Characteristics		Continuous, Numeric (Real)	
Number of Instances per Class		357, 212	

Both these data sets have only 2 classes, however the size of classes in both these data sets differ quite a bit.

7.1.1.5 Ionosphere

This radar data was collected by a system in *Goose Bay, Labrador*. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere. Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this data set are described by 2 attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal.

The details of the data-set are as follows:

Table 6: Data Set: Ionosphere

Area	Geophysics	**Data Set Characteristics**	Multivariate
Data Size	351	**Number of Classes**	2
Number of Attributes	34	**Missing Attribute Values**	None
Attribute Characteristics		Continuous, Numeric (Integer, Real)	
Number of Instances per Class		126, 225	

Though this data set is small and has only two classes, however the cluster sizes are quite different and the data is highly noisy.

7.1.1.6 Spambase

The *SPAM E-mail Database* was generated at the *Hewlett-Packard Labs*.

The "spam" concept is diverse: advertisements for products/web sites, make money fast schemes, chain letters, etc. The collection of spam e-mails came from postmasters and individuals who had filed spam. The collection of non-spam e-mails came from filed work and personal e-mails, and hence the word 'george' and the area code '650' are indicators

of non-spam. These are useful when constructing a personalized spam filter. One would either have to blind such non-spam indicators or get a very wide collection of non-spam to generate a general purpose spam filter.

The attributes of this data set represent features such as number of occurrences of known 'spam' words, frequency of word and character repetitions, frequency of capital letters, length of capital letters, etc.

The details of the data-set are as follows:

Table 7: Data Set: Spambase

Area	Computer Science	**Data Set Characteristics**	Multivariate
Data Size	4601	**Number of Classes**	2
Number of Attributes	57	**Missing Attribute Values**	Yes
Attribute Characteristics		Continuous, Numeric (Integer, Real)	
Number of Instances per Class		1813, 2788	

This is a large data set and the sizes of the clusters vary vastly. Moreover, the data is extremely noisy and has missing attribute values.

7.1.1.7 Mushroom

This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like *"leaflets three, let it be" for Poisonous Oak and Ivy*. The task is to classify the mushrooms into edible and poisonous. This is a popular categorical data set and has been popularly used in cluster analysis (Guha et al., 1999).

The details of the data-set are as follows:

This is a large data set where the sizes of the clusters are almost similar. However, has a lot of missing attribute values.

Table 8: Data Set: Mushroom

Area	Biology	**Data Set Characteristics**	Categorical
Data Size	8124	**Number of Classes**	2
Number of Attributes	22	**Missing Attribute Values**	Yes (2480)
Attribute Characteristics		Continuous, Numeric (Real)	
Number of Instances per Class		4208, 3916	

7.1.2 Synthetic Data Sets

To emphasize the effect of noise on different algorithms we also compare the algorithms on some synthetic data sets. We describe these synthetic data sets below.

*7.1.2.1 Syn*1

This data set is an instance of the example discussed in Chapter 3 and is described in Figure 17. The instance consists of 4 clusters, each having 10 points. Each point had a similarity of 0.9 to all points in it's own cluster. For each point in cluster 1 there is a corresponding point in cluster 3 to which it has a similarity 1 and vice versa. Similarly, for clusters 2 and 4. All other point pairs have a similarity value of 0. This data set satisfies the α-good neighborhood property for $\alpha = 1/n$. Note that the data is *not* in metric space.

The details of the data-set are as follows:

Table 9: Data Set: Syn1

Area	Synthetic	**Data Set Characteristics**	Non-Metric
Data Size	40	**Number of Classes**	4
Noise parameter α		$1/40$	
Number of Instances per Class		10, 10, 10, 10	

*7.1.2.2 Syn*2

This data set consists of 4 clusters, one for each quadrant as shown in Figure 30 which also describes the distance between points. We assume that size of B_1 or B_3 is no more than αn (for some α) and the size of B_1 is at least $2\alpha n$. Note that the *Syn2* is in Euclidean space and

satisfies the α-good neighborhood property for assumed value of α.

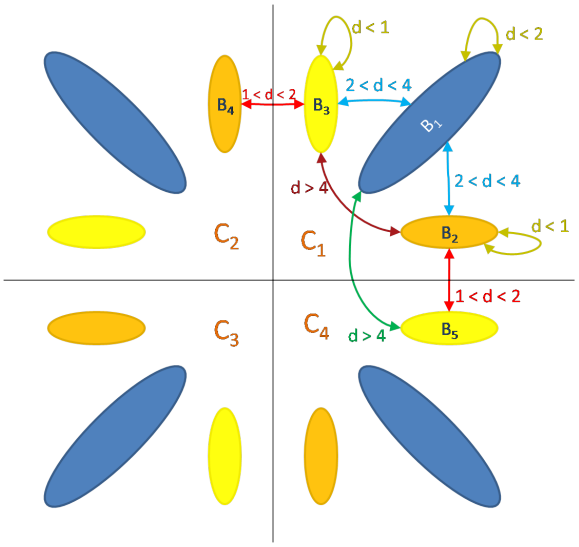


Figure 30: Syn2: Data lies in $2-D$ Euclidean space and each quadrant i represents a target cluster C_i. Each cluster consists of some stable points that have no noise (B_1) and some marginal points that have noise (B_2, B_3). The distances between points are defined in the figure.

The details of the data-set are as follows:

Table 10: Data Set: Syn2

Area	Synthetic	**Data Set Characteristics**	Metric
Data Size	64	**Number of Classes**	4
Noise parameter α		1/16	
Number of Instances per Class		16, 16, 16, 16	

7.1.2.3 Syn3

This data consists of 2 clusters, one for each plane as shown in Figure 31 which also describes the distance between points. Note that the data is in Euclidean space. We assume that size of each blob B_i is αn (for some α). Note that the *Syn2* is in Euclidean space and satisfies the α-good neighborhood property for assumed value of α.

The details of the data-set are as follows:

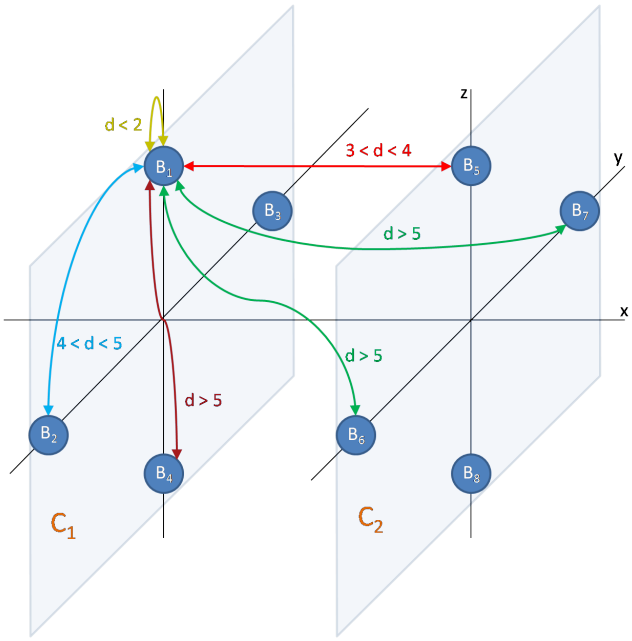


Figure 31: Syn3: Data lies in $3-D$ Euclidean space and the two YZ planes represent the two target clusters C_1 and C_2. Each cluster consists of 4 blobs (subset of points) equidistant from $(x, 0, 0)$. For each blob (B_1) in cluster C_1 there is another blob (B_5) in cluster C_2 to which all the points in that blob are closer than to points any other blob $(B_2,\ B_3,\ B_4)$ from it's own target cluster.

Table 11: Data Set: Syn3

Area	Synthetic	**Data Set Characteristics**	Metric
Data Size	64	**Number of Classes**	2
Noise parameter α		1/8	
Number of Instances per Class		32, 32	

7.2 Experimental Setup

To implement all the algorithms, *MATLAB* was used as the primary programming language. The *MATLAB* implementation is briefly described in Appendix D.

Before computing the similarity matrix, for each data set, several preprocessing steps were applied to sanitize the data.

1. For large data sets (like *Digits, Spambase, Mushroom, etc.*) where working on the entire data set was computationally prohibitive, a random sample of at most 1000 points was chosen *i.i.d.* from the data set and this sample data set was used in the

experiments.

2. Each of the d attributes was first normalized so that the data could be represented in $[0, 1]^d$ hypercube.

The similarity matrix generated from the data was then normalized to lie the allowed range of $[0, 1]$. This normalized similarity matrix was then passed to the algorithms as an input for optional use along with the data set.

All experiments were conducted on a 64 bit machine with $2GB$ RAM running Linux 2.6.

Note: For fairness of results, experiments were repeated several times on the same data over a range of values for input parameters for all algorithms accepting input parameters to get their best results.

7.3 Results

In this section we discuss the results of our experiments.

For all the data sets, the ground truth clustering is already known. The clustering computed by the algorithms was compared to this ground truth under the framework described in Chapter 3. The performance of each algorithm was computed using *Classification Error*.

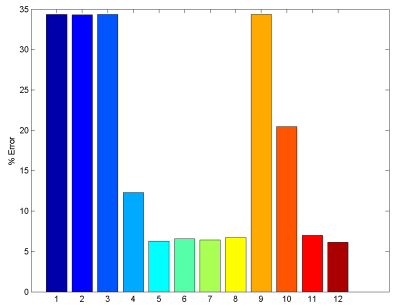
7.3.1 Transductive Setting

We discuss the results of running different algorithms in a transductive setting, *i.e.* the normal case where the algorithm uses all the points in the data set.

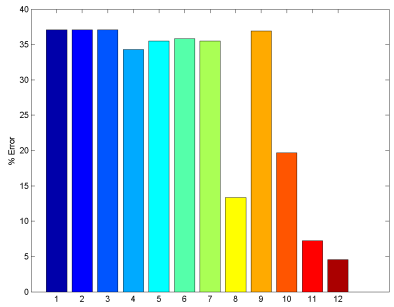
First we discuss the results of experiments conducted on various real-world data sets. Figures 32, 33, 34 and 35 show theses results.

It is easy to see from these figures that irrespective of the size of the data set *single linkage* performs extremely poorly on all the data sets.

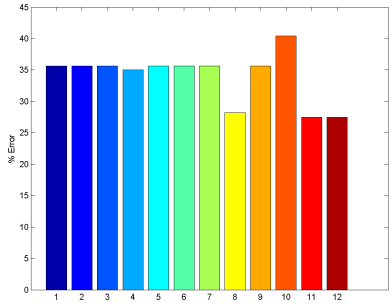
For smaller data sets like *Iris, Wine, Breast Cancer Wisconsin*, etc. where the data sets are not as noisy *complete linkage* does reasonably well and performs better than the *single*



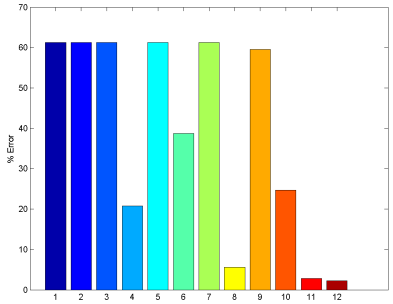
(a) Breast Cancer Wisconsin



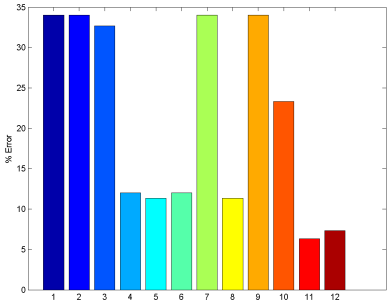
(b) Breast Cancer Wisconsin (Diagnostic)



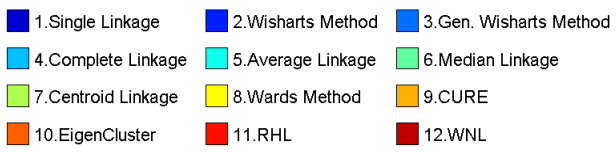
(c) Ionosphere



(d) Wine



(e) Iris

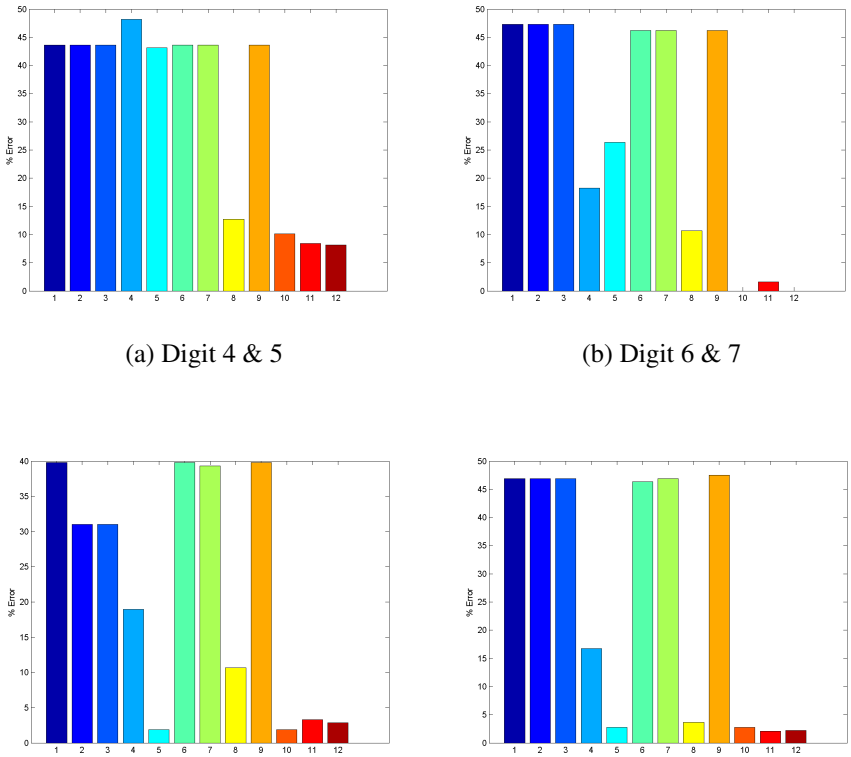


(f) legend

Figure 32: Classification Error of Algorithms for Small Data Sets. The y-axis in each case represents the % error.

linkage algorithm. However, in case of noisy data sets, it performs extremely poorly irrespective of the size of the data set, *e.g. Breast Cancer Wisconsin(Diagnostic), Ionosphere, Spambase*.

Average linkage performs well on few small data sets like *Digits(0 & 1), Digits (0 & 9)* where there is very less noise. However, as noise increases it can perform as badly as *single linkage* or *complete linkage* as seen in data sets *Wine, Ionosphere*, etc.



(a) Digit 4 & 5

(b) Digit 6 & 7

(c) Digits 0 & 1

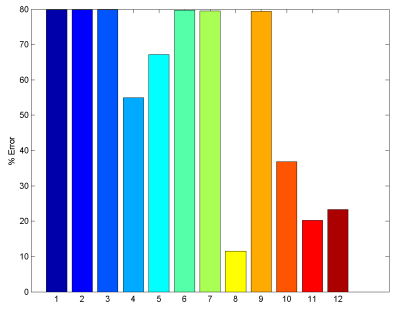
(d) Digits 0 & 9

(e) legend

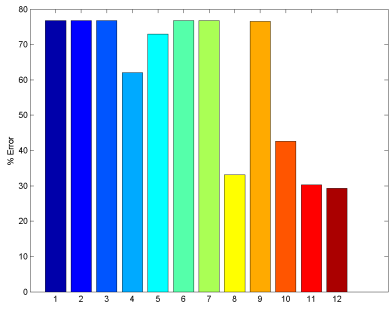
Figure 33: Classification Error of Algorithms for data set *Digit (Pairs)*. The *y*-axis in each case represents the % error.

Though *Centroid linkage* and *median linkage* algorithms perform better than *single linkage* in a few cases (*Iris, Breast Cancer Wisconsin*), however in most cases they perform as poorly as the *single linkage* algorithm irrespective of the size of the data sets.

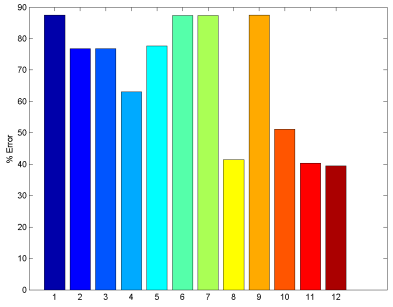
Ward's Linkage is perhaps the best among all standard linkage algorithms *(whenever applicable)* and consistently has much better performance than all other standard hierarchical algorithms.



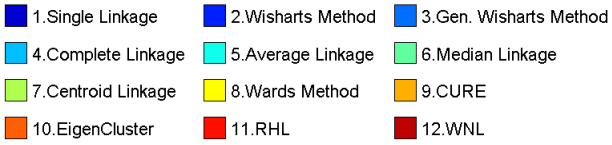
(a) Digit 6 & 7



(b) Digits 5,6,7,8 & 9



(c) All Digits



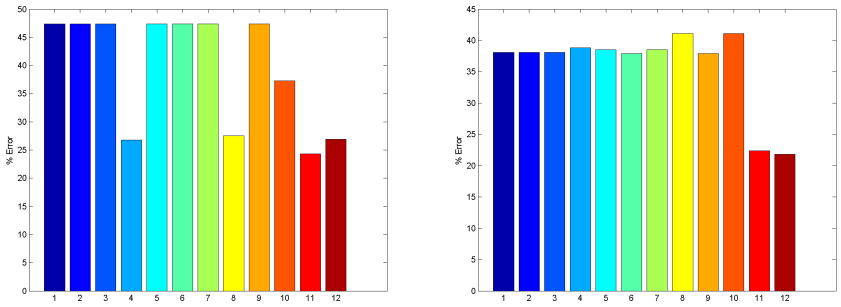
(d) legend

Figure 34: Classification Error of Algorithms for data set *Digits(larger subsets)* . The *y*-axis in each case represents the % error.

Moreover, we note that since most real world data sets have overlapping clusters, even the robust extensions of single linkage like the *Wishart's Method* (Wishart, 1969), the *(Generalized) Wishart's Method*(Chaudhuri & Dasgupta, 2010) and the *CURE* algorithm report extremely high error values very similar to standard linkage algorithms. Thus, even these

robust extensions are not as useful on real world data since they are not robust of one or the other structural property of data clusters as discussed in Chapter 3.

Though *EigenCluster* performs extremely well on some data sets like *Digits (0 & 1), (0 & 9), (4 & 5)*, etc. still, it is extremely inconsistent and reports extremely high errors even on some of the simpler data sets like *Iris, Breast Cancer Wisconsin*, etc. while failing miserably on others like *Spambase, Ionosphere*, etc.



(a) Mushroom

(b) Spambase

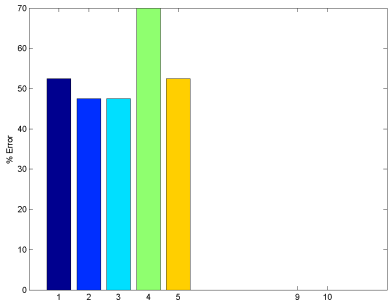
(c) legend

Figure 35: Classification Error of Algorithms for large data sets. The *y*-axis in each case represents the % error.

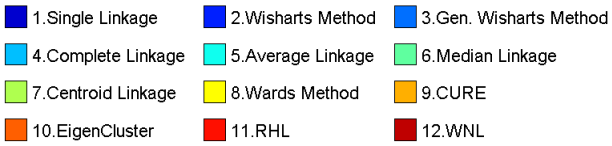
On the other hand, both our algorithms, *RHL* and *WNL*, have similar low error results on all data sets and consistently perform much better than all other algorithms irrespective of type or size of data set or the amount of noise present in the data set. This shows that both these algorithms are much more robust to various forms of noise present in the real-world data sets than other agglomerative algorithms.

For the *Synthetic* data set, experiments (Figure 36a) show that all standard linkage algorithms have an error greater than 50% on this data set, whereas both *RHL* and *WNL* have 0 error. Moreover, algorithms like *centroid linkage*, *median linkage* and *ward's method* are

not even applicable on this data set as it is not in metric space. This shows that not only are *RHL* and *WNL* robust to noise but also generic and easily applicable to any data for which pairwise similarity information can be computed.



(a) Syn1



(b) legend

Figure 36: Classification Error of Algorithms for synthetic data sets. The y-axis in each case represents the % error.

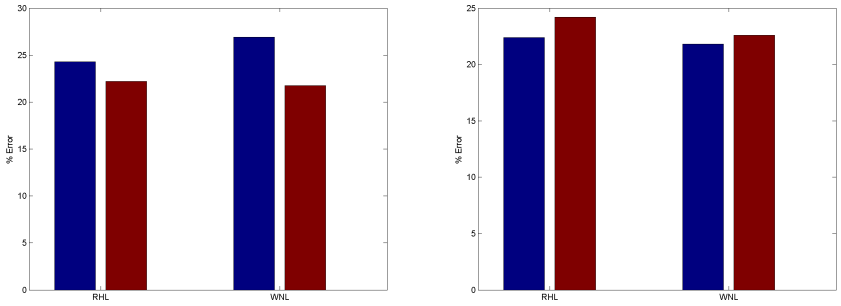
For $Syn2$, we know that all the linkage algorithms would merge all points in blobs B_2 with B_5 and points in B_3 with B_4 before they merge with the blob B_1. Considering the worst case where the sizes of blobs B_3 and B_2 is as high as possible, *i.e.* αn each, and the size of B_1 is as small as possible, *i.e.* $2\alpha n$, the error for all linkage based algorithms would be greater than 25% whereas *RHL* and *WNL* would cluster without error. For *Syn3*, we know that all the linkage algorithms would merge all points in blobs B_1 with B_5, B_2 with B_6, B_3 with B_7 and B_4 with B_8 before they merge blobs B_1, B_2, B_3, $B4$. Considering the worst case where the sizes of each blob is as high as possible, *i.e.* αn each, the error for all linkage based algorithms would be greater than 50% whereas *RHL* and *WNL* would cluster without error. The results for both $Syn2$ and $Syn3$ were also verified experimentally.

These experiments show that both our algorithms are much more robust to various

forms of noise present in the real-world data sets than other agglomerative algorithms and are much better alternatives to standard linkage algorithms for clustering data where certain amount of noise is expected to be present.

7.3.2 Inductive Setting

Now, we discuss the results of running both our algorithms, *RHL* and *WNL* in the inductive setting, *i.e.* first, a small subset of points is randomly chosen from a much larger instance space to generate a hierarchy and then the rest of the points are inserted into this hierarchy resulting in a hierarchy over the entire instance space.



(a) Mushroom

(b) Spambase

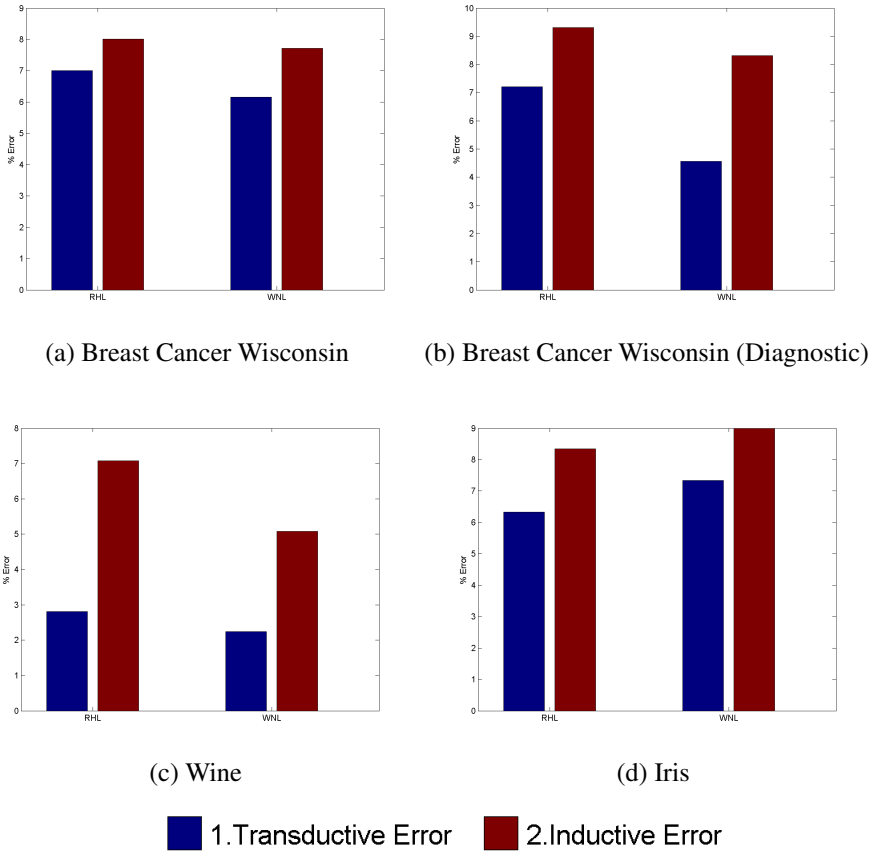
(c) legend

Figure 37: Comparison of Classification Error computed on the initial sample and on the complete data set after the inductive step. The y-axis in each case represents the % error.

Note that experiments on other algorithms were not conducted in the inductive setting as there are no known ways of extending the standard linkage algorithms to an inductive setting. On the other hand, our algorithm can be easily extended to an inductive setting and requires only a small random sample which is independent on n and depends only on the noise parameter α and the confidence parameter δ.

For data sets such as *Mushroom, Spambase and Digits*, which have a large number of

points, it was prohibitive for us to run the transductive versions of algorithms due to limited memory, but we were easily able to run the inductive versions of our algorithms by first running the algorithm on a random sample of small number of points (*e.g.* range [500, 1000]) and then inserting the rest of the points to the generated hierarchy. Experiments show that the increase in error in the inductive phase is minimal, where error even decreases in some cases, as compared to the error computed on the sample (Figure 37). Thus, the error over the entire data set is almost the same as that on the initial sample.



(a) Breast Cancer Wisconsin

(b) Breast Cancer Wisconsin (Diagnostic)

(c) Wine

(d) Iris

(e) legend

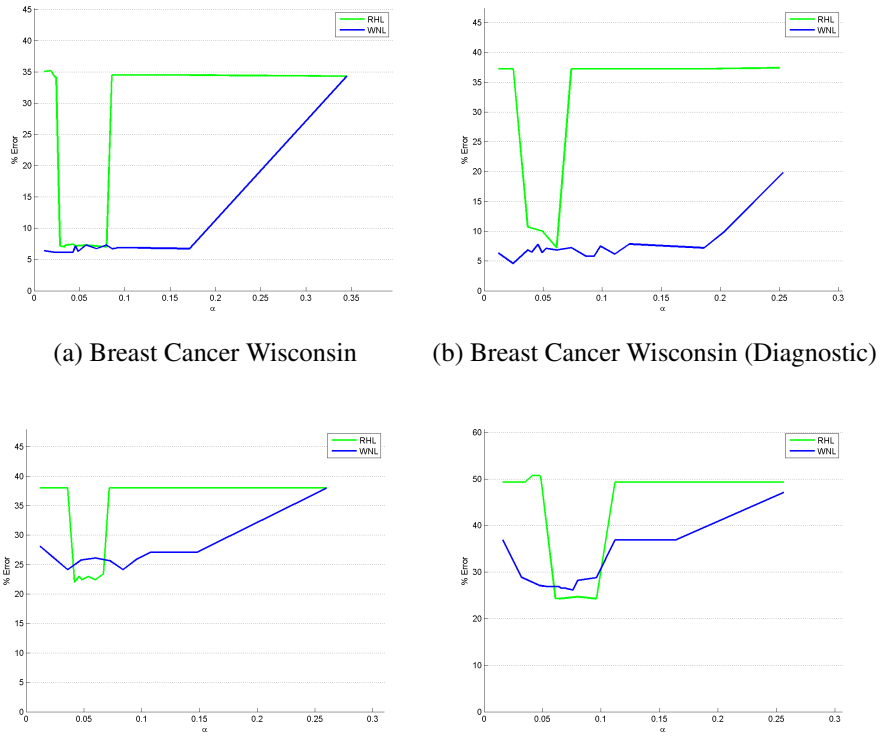
Figure 38: Comparison of Classification Error in the Inductive Setting and the Transductive Setting. The *y*-axis in each case represents the % error.

Also, for data sets such as *Iris, Wine, Breast Cancer Wisconsin*, for which could compute run the algorithm in the transductive setting, a comparison with the inductive setting showed that there was only a small increase in error in the inductive version. Figure 38 shows the results.

These results indicate the usefulness of inductive version of our algorithm as a quicker and simpler alternative when analysis over the complete data set is difficult.

7.3.3 Variation in Input Parameters

To characterize noise, our algorithms require extra parameters as input whereas standard linkage algorithms do not require any input parameters. Algorithm *RHL* requires two parameters α and ν as input whereas *WNL* only requires the parameter α as input. Though *RHL* takes two different parameters as input, on close observation of the algorithm, we can see that both the parameters are always used together as the additive term $(\alpha + \nu)$. We can see this term as just a single input parameter α' and vary it linearly. This makes the analysis by varying input parameters much simpler and easier to evaluate.



(a) Breast Cancer Wisconsin

(b) Breast Cancer Wisconsin (Diagnostic)

(c) Spambase

(d) Mushroom

Figure 39: Performance of *RHL* and *WNL* vs. Variation in α.

As discussed in Chapter 5, given a new data set, though there may be ways to estimate the value of parameter α, there is no clear way to compute it's exact value beforehand.

As observed in Figure 39, this is a tangible issue with the *RHL* algorithm as it is not very robust to varying values of α and gives results of low error over a small range of the values of α.

However, this is not a limitation for the *WNL* algorithm as experiments with varying values of α reveal that it does not require the exact value of α to be known beforehand as it does extremely well over a wide range of values of α (Figure 39). Since the desired values of α for *WNL* is in range $[0, 0.25]$ it is extremely easy to land in the right range with only a constant number of runs of the algorithm. Thus, even on new data sets, we can still use the algorithm to generate a hierarchy with low error. Thus, as shown in Figure 39, *WNL* is much less sensitive to change in α as compared to *RHL*.

7.4 Discussion

In this chapter we do a systematic experimental analysis of several hierarchical clustering algorithms evaluate their robustness by comparing their performance by comparing their results on a wide variety of *synthetic* and *real-world* data sets(Frank & Asuncion, 2010) and showed that our algorithms *RHL* and *WNL* based on the good neighborhood property, consistently perform much better than other hierarchical algorithms and are therefore much more robust to various forms of noise present in the data.

Further, we showed the efficacy of the inductive versions of our algorithms as a quicker and simpler alternative when analysis over the complete data set is difficult or even prohibitive as the increase in error on using inductive version of our algorithms is insignificant.

Also, we discussed the robustness of our algorithms with respect to input parameters and observed that though *RHL* is not as robust to varying values of input parameters, *WNL* on the other hand performs well on a wide range of possible values of the input parameters and hence is extremely robust to incorrect estimates of the input parameters.

Since *WNL* is provably faster that *RHL*, is shown experimentally to perform consistently comparably to *RHL* and is more robust to parameter tuning, it proves to be an extremely useful alternative to the *RHL* algorithm where data sets satisfy similar structural properties.

CHAPTER VIII

CONCLUSION

In this work we propose and analyze two new robust algorithms, *RHL* and *WNL* for bottom-up agglomerative clustering. We show that our algorithms can be used to cluster accurately in cases where the data satisfies a number of natural properties and where the traditional agglomerative algorithms fail. In particular, if the data satisfies the (α, ν)-good neighborhood property, the *RHL* will be successful in generating a hierarchy such that the target clustering is a pruning of that hierarchy, whereas *WNL* will be successful if the data satisfies the $(\alpha, 0)$-good neighborhood property.

We also show how to extend our algorithms, with similar correctness guarantees for robustness, to an inductive setting where the given data is only a small random sample of the entire data set. This contrasts sharply with the fact that there are no known ways of extending the standard linkage algorithms to the inductive setting and thus have to be run over the entire data set. Moreover, the inductive versions of our algorithms require only a small random sample which is independent on the size of instance space and depends only on the noise parameters α, ν and the confidence parameter δ.

We further present a comprehensive experimental analysis of most popular and widely used hierarchical clustering algorithms like *Standard Linkage algorithms, (Generalized)Wishart's Method, CURE, EigenCluster*, etc. and compare their results with our algorithms *RHL* and *WNL* on a wide variety of synthetic and real-world data sets and show that both our algorithms consistently perform much better than other hierarchical algorithms and are much more robust to various forms of noise present in the data. Further, we show experimentally the efficacy of the inductive versions of our algorithms as a quicker and simpler alternative when analysis over the complete data set is difficult or even prohibitive.

We observe that *standard linkage algorithms* succeed under strict separation but fail badly under the *good neighborhood* property. Moreover, we show experimentally, that robust extensions of the standard linkage algorithms like *Wishart's Method, CURE*, etc. perform badly on real world data and are not good very alternatives to standard linkage algorithms for noisy real-world data.

We note that while being a robust algorithm, *RHL* is not completely agglomerative and is algorithmically quite complex and computationally intensive due to it's running time of $O(n^{\omega+1})$. Moreover, it requires two parameters α, ν as input. On the other hand, though, *WNL* can be proven theoretically only for the simpler α-good neighborhood property, it turns out to be a more practical algorithm as compared to *RHL* as it is provably faster, algorithmically simpler, completely agglomerative in nature, requires only a single input parameter and can be shown experimentally to perform comparably to *RHL* and to be much more robust to parameter tuning, *i.e.* incorrect estimates of the input parameter α. Since *WNL* is provably faster that *RHL*, is shown experimentally to perform consistently comparably to *RHL* and is more robust to parameter tuning, it proves to be an extremely useful alternative to the *RHL* algorithm where data sets satisfy similar structural properties and an ideal clustering technique in the presence of noise.

8.1 Future Work

It would be interesting to see if our algorithmic approach can be shown to work for other natural properties. For example, it would be particularly interesting to analyze a relaxation of the *max stability* property which was shown to be a necessary and sufficient condition for *single linkage*, or the *average stability* property which was shown to be a sufficient condition for average linkage (as discussed in Chapter 3).

A major drawback of *WNL* is that it is still order of n slower than the fastest versions of standard linkage algorithms. It would be great to be able to bridge this gap either by improve the running times of our algorithms or developing new algorithms that can bridge

this gap while still satisfying the good neighborhood property. For example, possibly find hybrid algorithms that use a more complex neighborhood property in the initial phase and then revert to simpler property as the sizes of the clusters increase thereby reducing the amortized running time.

Defining new properties to better characterize real-world noise or properties using which algorithms like *centroid linkage* and other robust extensions like *Wishart's Method, CURE*, etc. that would allow us to formally compare the robustness of these algorithms with our algorithms is another interesting line of work.

Currently our algorithms use the un-weighted version of neighborhood weights, *i.e.* given three points $x, y, z \in S$, neighbor $z's$ contributes a weight w_z to x and y is defined as

$$w_z(x, y) = \left\{ \begin{array}{ll} 1 & \text{if } z \in N_t(x) \text{ and } z \in N_t(y) \\ 0 & \text{otherwise} \end{array} \right\}$$

where $N_t(x)$ is the set of t nearest neighbors of x. It would be interesting to see if it is possible to develop algorithms with formal correctness guarantees (possibly using new neighborhood properties) where

$$w_z(x, y) = f(\mathcal{K}(x, z), \mathcal{K}(y, z))$$

where f is some arithmetic function returning a value in range $[0, 1]$.

Another interesting line of work would be to develop and analyze possible inductive extensions of standard linkage algorithms.

APPENDIX A

CORRECTNESS RESULTS FOR STANDARD LINKAGE ALGORITHMS

Below we provide the proofs for the theorems discussed in Chapter 3

Theorem 15. *Let $\mathcal{K}$ be a symmetric similarity function satisfying the strict separation property. Then we can efficiently construct a tree using single linkage algorithm such that the ground-truth clustering is a pruning of this tree.*

Proof: We can use the single linkage algorithm (*i.e.* Kruskal's algorithm) to produce the desired tree. We begin with n clusters of size 1 and at each step we merge the two clusters C,C' maximizing $\mathcal{K}_{max}(C, C')$. This procedure maintains the invariant that at each step the current clustering is laminar with respect to the ground-truth (every cluster is either contained in, equal to, or a union of target clusters). In particular, if the algorithm merges two clusters C and C', and C is strictly contained in some cluster C_r of the ground truth, then by the strict separation property we must have $C \subset C_r$ as well. Since at each step the clustering is laminar with respect to the target, the target clustering must be a pruning of the final tree. ■

Theorem 16. *For a symmetric similarity function $\mathcal{K}$, Property 3 is a necessary and sufficient condition for single linkage to produce a tree such that the ground truth clustering is a pruning of this tree.*

Proof: We first show that if $\mathcal{K}$ satisfies Property 3, then the single linkage algorithm will produce a correct tree. By induction we maintain the invariant that at each step the current clustering is laminar with respect to the ground-truth. In particular, if some current cluster

$A \subset C_r$ for some target cluster C_r is merged with some other cluster B, Property 3 implies that B must also be contained within C_r. In the other direction, if the property is not satisfied, then there exist A, A' such that $\mathcal{K}_{max}(A, C_r \backslash A) \leq \mathcal{K}_{max}(A, A')$. Let y be the point not in C_r maximizing $\mathcal{K}(A, y)$. Let us now watch the algorithm until it makes the first merge between a cluster C contained within A and a cluster C'disjoint from A. By assumption it must be the case that $y \in C'$, so the algorithm will fail. ■

Theorem 17. *Let $\mathcal{K}$ symmetric similarity function $\mathcal{K}$ satisfying strong stabilityThen average linkage algorithm constructs a binary tree such that the ground truth clustering is a pruning of this tree.*

Proof: We prove correctness by induction. In particular, assume that our current clustering is laminar with respect to the ground truth clustering (which is true at the start). That is, for each cluster C in our current clustering and each C_r in the ground truth, we have either $C \subseteq C_r$, *or* $C_r \subseteq C$ *or* $C \cap C_r = \phi$. Now, consider a merge of two clusters C and C'. The only way that laminarity could fail to be satisfied after the merge is if one of the two clusters, say, C', is strictly contained inside some ground-truth cluster C_r (so, $C_r - C' \neq \phi$) and yet C is disjoint from C_r.

Now, note that by strong stability property, $\mathcal{K}(C', C_r - C') > \mathcal{K}(C', x)$ for all $x \notin C_r$, and so in particular we have $\mathcal{K}(C', C_r - C') > \mathcal{K}(C', C)$. Furthermore, $\mathcal{K}(C', C_r - C')$ is a weighted average of the $\mathcal{K}(C', C'')$ over the sets $C'' \subseteq C_r - C'$ in our current clustering and so at least one such C'' must satisfy $\mathcal{K}(C', C'') > \mathcal{K}(C', C)$. However, this contradicts the specification of the algorithm, since by definition it merges the pair C, C' such that $\mathcal{K}(C', C)$ is greatest. ■

Theorem 18. *Let $\mathcal{K}$ symmetric similarity function $\mathcal{K}$ satisfying weak stabilityThen average linkage algorithm constructs a binary tree such that the ground truth clustering is a pruning of this tree.*

Proof: We prove correctness by induction. In particular, assume that our current clustering is laminar with respect to the ground truth clustering (which is true at the start). That is, for each cluster C in our current clustering and each C_r in the ground truth, we have either $C \subseteq C_r$, *or* $C_r \subseteq C$ *or* $C \cap C_r = \phi$. Now, consider a merge of two clusters C and C'. The only way that laminarity could fail to be satisfied after the merge is if one of the two clusters, say, C', is strictly contained inside some ground-truth cluster $C_{r'}$ and yet C is disjoint from $C_{r'}$.

We distinguish a few cases. First, assume that C is a cluster C_r of the ground truth. Then by definition, $\mathcal{K}(C', C_{r'} - C') > \mathcal{K}(C', C)$. Furthermore, $\mathcal{K}(C', C_{r'} - C')$ is a weighted average of the $\mathcal{K}(C', C'')$ over the sets $C'' \subseteq C_{r'} - C'$ in our current clustering and so at least one such C'' must satisfy $\mathcal{K}(C', C'') > \mathcal{K}(C', C)$. However, this contradicts the specification of the algorithm, since by definition it merges the pair C, C' such that $\mathcal{K}(C', C)$ is greatest. Second, assume that C is strictly contained in one of the ground-truth clusters C_r. Then, by the weak stability property, either $\mathcal{K}(C, C_r - C) > \mathcal{K}(C, C')$ or $\mathcal{K}(C', C_{r'} - C') > \mathcal{K}(C, C')$. This again contradicts the specification of the algorithm as in the previous case. Finally, assume that C is a union of clusters in the ground-truth $C_1, ...C_{k'}$. Then by definition, $\mathcal{K}(C', C_{r'} - C') > \mathcal{K}(C', C_i)$, for $i = 1, ...k'$, and so $\mathcal{K}(C', C_{r'} - C') > \mathcal{K}(C', C)$. This again leads to a contradiction as argued above. ■

APPENDIX B

CLUSTERING VALIDITY

There exists a large variety of clustering algorithms, each having certain advantages but also certain drawbacks. Typical questions that arise in this context while looking for an optimal algorithm are:

- How similar are the two clusterings for a given data set?
- If a ground-truth clusterings is available: How close is the clustering solution to the optimal one?
- How similar are the clusterings of two different algorithms?
- Which algorithm is better?
- Is the algorithm sensitive to the order of the data, *i.e.* can another order of the data result in a very different clustering?
- Is the algorithm sensitive to noise, *i.e.* can small changes in the data entail large changes in the clustering?

Comparing clusterings plays an important role in the evaluation of clustering algorithms. If the algorithm is not completely deterministic (e.g. the result may depend on initial conditions), the operation may be repeated several times, and the resulting distances to the correct clustering may be averaged to yield the algorithm's average performance. Moreover, this average may be compared to another average distance obtained in the same way for another algorithm. Thus, we require some *measure* of the similarity between two clusterings or for their *distance* or dissimilarity. In practice, distances between clusterings are

subject to addition, subtraction and other complex operations. We need a distance measure that is valid under such operations.

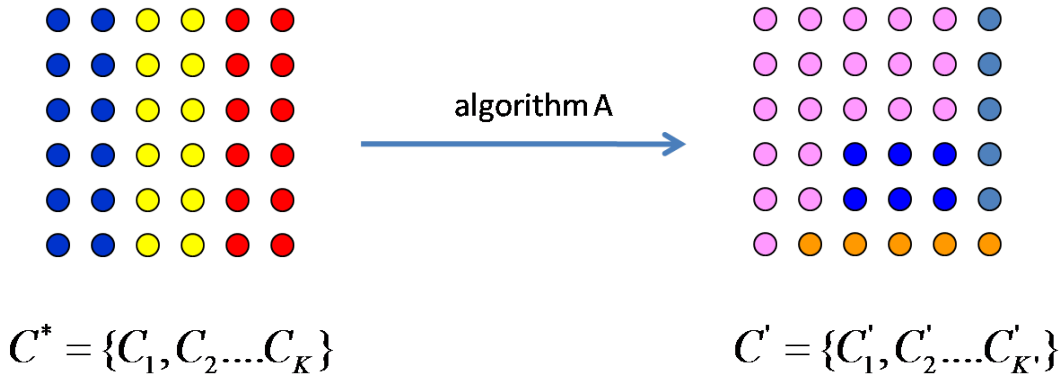


Figure 40: Ground-truth clustering Vs. Clustering Solution from algorithm A

B.1 Requirements for a Good Measure

Before formalizing the axioms defining a *good* distance measure d for comparing clusterings, we discuss some properties that are desirable. The distance measure d should desirably have the following properties:

1. Can be applied to any two clusterings in $\mathcal{P}(D)$.

2. Should not have any constraints on the structure of the clusterings or on their relation, *i.e.* it should make no assumptions on

   (a) Cluster Size.

   (b) Number of Clusters (C and C' can have different number of clusters)

   (c) Dependencies between C and C'.

   The result should not be affected by assumptions on the clusterings which, in general, are not fulfilled. Measures that make such assumptions on the structure of the clusterings and their relation, cannot yield reliable results.

3. Independence from the number of clusters.

   Random clusterings can have a high similarity just because they have a large number of clusters. In order to avoid this we need independence from the number of clusters.

4. n-invariance.

   We need to make sure that clusterings of different data sets with different sizes are on the same scale. This is important for comparing distance values of clustering pairs of different data sets. When we want to compare clustering algorithms, this allows us to calculate the *mean error* for different algorithms, which is the average of the distances to the optimal clustering for different data sets.

5. Metric. This allows us to extend the notion of a straight line from Euclidean space to a metric space and compute distance between clusterings as sum of line segments. The properties of a metric mainly the symmetry and the triangle inequality make a criterion more understandable. The triangle inequality tells us that if two elements of a metric space (*i.e.* clusterings) are close to a third they cannot be too far apart from each other. This property is extremely useful in designing efficient data structures and algorithms. With a metric, one can move from simply comparing two clusterings to analyzing the structure of large sets of clusterings.

B.1.1 Notation

Given a *data set* D containing n points, a clustering C is a partition of D into non-empty, disjoint sets $\{C_1, C_2, \ldots, C_K\}$ called *clusters* such that

$$C_i \cap C_j = \phi \quad and \quad \bigcup_{i=1}^{K} C_k = D \tag{21}$$

The number of points in the cluster C_i is denoted by n_i such that $n_i \geq 1$ and

$$n = \sum_{i=1}^{K} n_i$$

The fraction of points in the cluster C_i is denoted by p_i where

$$p_i = \frac{n_i}{n}$$

This gives the probability that a randomly chosen point in the data set D belongs to cluster C_i.

A second clustering of S is denoted by $C' = \{C'_1, C'_2, \ldots, C'_{K'}\}$, where n'_j denotes the size of cluster C'_j.

The number of points of intersection of cluster $C_i \in C$ and $C'_j \in C'$ is denoted by n_{ij} where

$$n_{ij} = |C_i \cap C'_j| \tag{22}$$

The fraction of points of intersection of cluster $C_i \in C$ and $C'_j \in C'$ is denoted by p_{ij} where

$$p_{ij} = \frac{n_{ij}}{n} \tag{23}$$

This gives the probability that a randomly chosen point in the data set D belongs to clusters $C_i \in C$ and $C'_j \in C'$.

To compare C and C' we a distance measure $d(C, C')$ that measures how different two clusterings are. Alternatively we may use similarity is as a measure, which measures how similar two clusterings are.

All clusterings of a finite set D can be represented as the nodes of a graph, In this graph an edge between C, C' will be present if C' is obtained by splitting a cluster of C into two parts.

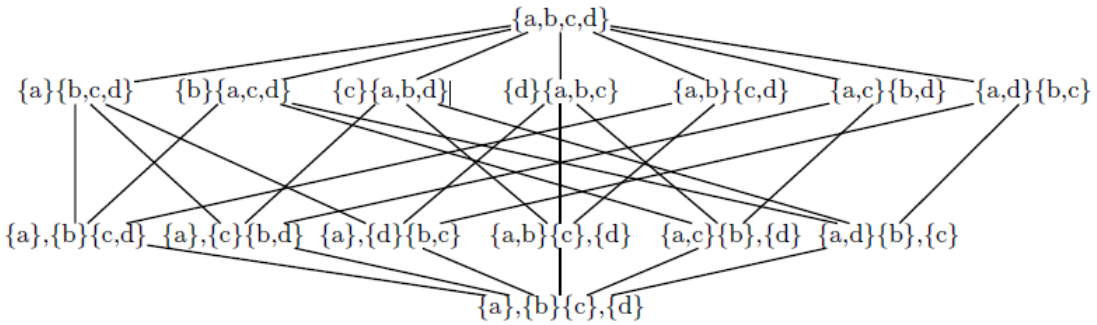


Figure 41: The Lattice $\mathcal{P}(D)$ for $D = \{a, b, c, d\}$.

The set of all clusterings of a dataset D forms a lattice called the lattice of partitions

$\mathcal{P}(D)$. where the graph represents a *Hasse diagram* (Figure 41) of this lattice. At the top of the diagram is the clustering with just one cluster, denoted by $\hat{1} = \{D\}$. At the bottom is $\hat{0} = \{\{1\}, \{1\}, \ldots, \{n\}\}$. the clustering with n clusters each containing a single point.

Intuitively, distance measure $d(C, C')$ represents sum of distances along edges of the lattice.

B.1.2 Properties for Distance Measure

We give formal definitions various properties related with distance measures.

Additivity w.r.t. Refinement (AR) If C' is obtained from C by splitting one or more clusters, then we say that C' is a *refinement* of C. A distance d is *additive w.r.t refinement* if and only if for any clusterings C, C', C'' such that C' is a refinement of C and C'' a refinement of C', we have

$$d(C, C'') = d(C, C') + d(C', C'')$$

This means that the distance $d(C, C'')$ is a sum of the distances corresponding to the two successive refinements that transform C into C''. Cluster splitting corresponds to taking down-ward steps in the Hasse diagram.

Additivity w.r.t. Join (AJ) The join of two clusterings is the clustering formed from all the nonempty intersections of clusters from C with clusters from C'. It is defined as

$$C \times C' = \{C_i \cap C'_j | C_i \in C, C'_j \in C', C_i \cap C'_j \neq \phi\}$$

A distance d is *additive w.r.t to the join* if and only if for any clusterings C, C'

$$d(C, C') = d(C, C \times C') + d(C', C \times C')$$

This property is relevant for clusterings C, C' which are not a refinement of each other. One can think of obtaining such a C' from C by a series of cluster splits (downward steps along

the lattice edges) followed by a set of cluster mergers (upward steps in the lattice). Note that usually there are several possible paths between two clusterings; the sum is the same no matter what path is taken.

Convex Additivity (CA) Let C', C'' be two refinements of C. Let C'_i and C''_i be the partitionings induced by C' and C'' on $C_i \in C$. Then d satisfies *convex additivity* if

$$d(C', C'') = \sum_{i=1}^{K} p_i d(C'_i, C''_i)$$

This implies that if only some cluster(s) of C are changed to obtain C', then $d(C, C')$ depends only on the affected clusters, and is independent on how the unaffected part of D is partitioned.

These properties are important as splitting a cluster, forming a clustering by taking the union of two clusterings on two subsets of the data, and merging two clusters are (sequences of) elementary and intuitive operations that one can do on clusterings. We can think of changing a clustering C into C' by applying these operations one after the other. If the distance d satisfies *AV, CA and AJ*, then d will measure the change between C and C' as a sum of elementary changes, each corresponding to one step.

n-Invariance Distance measure d is n-invariant if it only depends on p_i and p_{ij} and not on n, n_i, n_{ij}. *i.e.* it can computed entirely using the normalized confusion matrix P. See Figure 42.

Metric Distance measure d is a metric if it satisfies

1. Positivity: $d(C, C') \geq 0$.
2. Symmetry $d(C, C') = d(C', C)$.
3. Identity of Indiscernible: $d(C, C') = 0$ if and only if $C = C'$.
4. Triangle Inequality: $d(C, C'') \leq d(C, C') + d(C', C'')$.

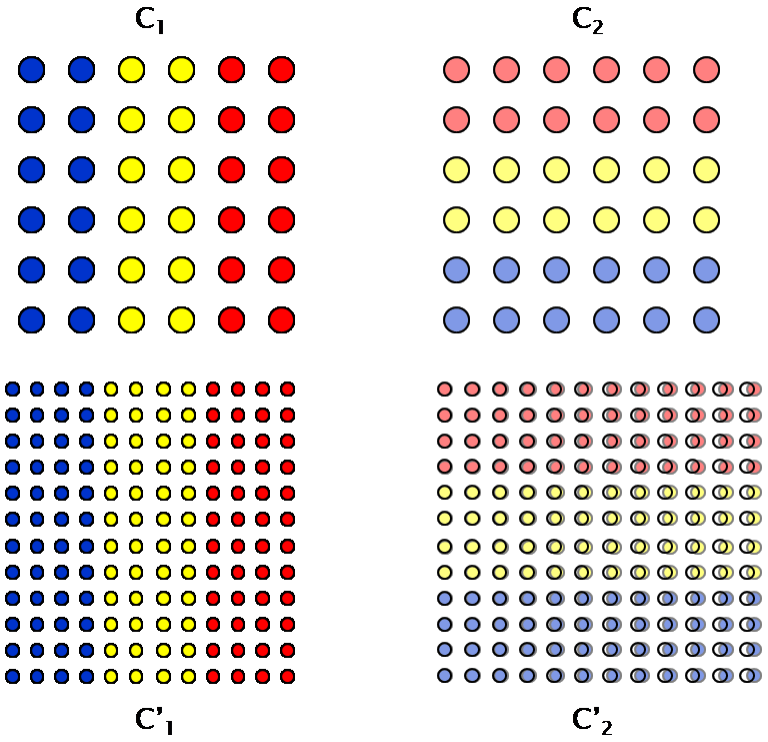


Figure 42: n-invariance. Here $d(C_1, C_2) = (C'_1, C'_2)$

If d is a metric, we can extend the notion of a straight line from Euclidean space to a metric space.

B.1.3 Axioms

We formally define some axioms (Meilă, 2005) to cover some of the quantitative requirements for a distance measure.

$\mathcal{A}1$ **Symmetry:** For any two clusterings $C, C' \in \mathcal{P}(D)$

$$d(C, C') = d(C', C) \tag{24}$$

$\mathcal{A}2$ **Additivity w.r.t. Refinement:** For any clustering $C \in \mathcal{P}(D)$

$$d(\hat{0}, C) + d(C, \hat{1}) = d(\hat{0}, \hat{1}) \tag{25}$$

$\mathcal{A}2.D$ For any clustering $C \in \mathcal{P}(D)$

$$d(\hat{1}, C) = \frac{1}{2}\left[1 - \frac{\max_k n_k}{n}\right] \tag{26}$$

$\mathcal{A}3$ **Additivity w.r.t. Join:** For any two clusterings $C, C' \in \mathcal{P}(D)$

$$d(C, C') = d(C, C \times C') + d(C', C \times C') \tag{27}$$

$\mathcal{A}4$ **Convex Additivity:** Let C' be a refinements of C. Let C'_i be the partitioning induced by C' on $C_i \in C$. Then

$$d(C, C') = \sum_{i=1}^{K} p_i d(\hat{1}_{n_k}, C'_i) \tag{28}$$

where $\hat{1}_{n_k}$ is the $\hat{1}$ clustering of the dataset C_k containing n_k points.

$\mathcal{A}4.M$

$$d(C, C') = \sum_{i=1}^{K} p_i^2 d(\hat{1}_{n_k}, C'_i) \tag{29}$$

$\mathcal{A}5$ **Scale:** Let C_K^U denote the *uniform* clustering, *i.e.* a clustering with K equal clusters. If C_K^U exists for a given data set D, then

$$d(\hat{1}, C_K^U) = \log K \tag{30}$$

$\mathcal{A}5.D$

$$d(\hat{1}, C_K^U) = 1 - \frac{1}{K} \tag{31}$$

Intuitively, axioms $\mathcal{A}2$ and $\mathcal{A}3$ describe the geometric properties of a distance measure ,*i.e.* the clustering is aligned with the lattice of partitions. Axiom $\mathcal{A}2$ is a weak version of the AR property. Axioms $\mathcal{A}4$ and $\mathcal{A}5$ set the scale of d and in particular its logarithmic growth rate.

The additivity axiom $\mathcal{A}4.M$ has several consequences. First, it shows that d is local, *i.e.* changes inside one cluster depend only on the relative size of that cluster and of the nature of the change, and are not affected by how the rest of the data set is partitioned. But the union of two or more data sets shrinks distances (because of the weighting with the squares of the proportions), a rather counterintuitive behavior.

B.2 Confusion Matrix

We define two matrices that are used by every measure for their computation.

For two clusterings $C = \{C_1, C_2, \ldots, C_K\}, C' = \{C'_1, C'_2, \ldots, C'_{K'}\}$, the *confusion matrix* M is defined as a $K \times K'$ contingency table (Figure 43) where the entry in the i_{th} row and j_{th} column represents n_{ij}, *i.e.* the number of common points between clusters $C_i \in C$ and $C'_j \in C'$.

C\C'	C_1'	C_2'		C_K'	Sums
C_1	n_{11}	n_{12}	...	$n_{1K'}$	n_1
C_2	n_{21}	n_{22}	...	$n_{2K'}$	n_2
...	...	...	$n_{kk'}$	...	...
C_K	n_{K1}	n_{K2}	...	$n_{KK'}$	n_K
Sums	n_1'	n_2'		n_K'	

Figure 43: Confusion Matrix M

Note that

$$\sum_{i=1}^{K} n_{ij} = n'_j \;\; \& \;\; \sum_{j=1}^{K'} n_{ij} = n_i \;\; \& \;\; \sum_{i=1}^{K} \sum_{j=1}^{K'} n_{ij} = n$$

A *normalized version of the confusion matrix* P can be obtained by dividing the confusion matrix throughout by M. *i.e.* $P = \frac{M}{n}$. Thus, the entry in the i_{th} row and j_{th} column of P represents p_{ij}, *i.e.* the fraction of common points between clusters $C_i \in C$ and $C'_j \in C'$ or the probability that a randomly chosen point in the data set D belongs to clusters $C_i \in C$ and $C'_j \in C'$. Note that

$$\sum_{i=1}^{K} p_{ij} = p'_j \;\; \& \;\; \sum_{j=1}^{K'} p_{ij} = p_i \;\; \& \;\; \sum_{i=1}^{K} \sum_{j=1}^{K'} p_{ij} = 1$$

B.3 Methods based on Counting Pairs

A very intuitional approach to comparing clusterings is counting pairs of points that are *classified* in the same way in both clusterings, *i.e.* pairs of points of D that are either in the

same cluster or in different clusters in both clusterings respectively.

A pair of points from D can fall under one of four cases described below (Figure 44):

N_{11}: #point pairs that are in the same cluster under both C and C'.

N_{00}: #point pairs in different clusters under both C and C'.

N_{10}: #point pairs in the same cluster under C but not under C'.

N_{01}: #point pairs in the same cluster under C' but not under C.

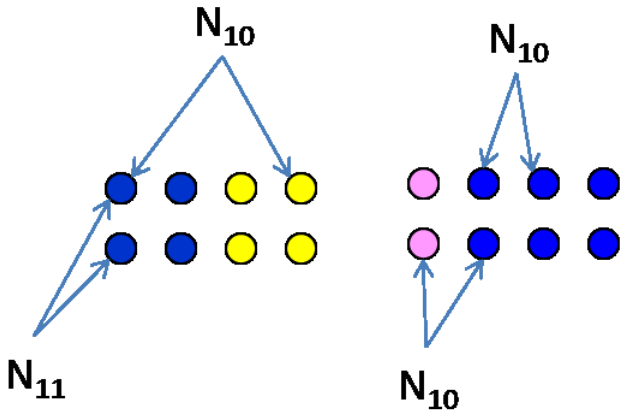


Figure 44: Point Pairs

Since the total number of point pairs is $\binom{n}{2}$, we get

$$N_{11} + N_{00} + N_{10} + N_{01} = \binom{n}{2}$$

We can obtain $N_{ij}, i, j = \{0, 1\}$ from the confusion matrix $M_{KK'}$ as follows.

$$\begin{aligned}
N_{11} &= \sum_{i,j} \binom{n_{ij}}{2} \\
N_{10} &= \sum_{i} \binom{n_i}{2} - \sum_{i,j} \binom{n_{ij}}{2} \\
N_{01} &= \sum_{j} \binom{n'_j}{2} - \sum_{i,j} \binom{n_{ij}}{2} \\
N_{00} &= \binom{n}{2} - \left(\sum_{i} \binom{n_i}{2} + \sum_{j} \binom{n'_j}{2} - \sum_{i,j} \binom{n_{ij}}{2} \right)
\end{aligned}$$

B.3.1 Chi Squared Coefficient

The most ancient measures for comparing clusterings were originally developed for statistical issues. *Chi Squared Coefficient* is one of the most well-known measures of this kind. It was suggested in 1900 by Pearson for testing independence in a bivariate distribution, not for evaluating association (which, in the context of clustering, corresponds to evaluating similarity). It is defined as

$$\chi(C, C') = \sum_{i=1}^{K} \sum_{j=1}^{K'} \frac{(n_{ij} - E_{ij})}{E_{ij}} \tag{32}$$

where

$$E_{ij} = \frac{n_i n_j}{n}$$

However, this method assumes independence of the two clusterings, which is not valid in general.

B.3.2 Wallace Criteria

(Wallace, 1983) proposed the two $\mathcal{W}_I, \mathcal{W}_{II}$ defined as:

$$\mathcal{W}_I(C, C') = \frac{N_{11}}{N_{11} + N_{10}} = \frac{\sum_{i,j} \binom{n_{ij}}{2}}{\sum_i \binom{n_i}{2}} \tag{33}$$

$$\mathcal{W}_{II}(C, C') = \frac{N_{11}}{N_{11} + N_{01}} = \frac{\sum_{i,j} \binom{n_{ij}}{2}}{\sum_j \binom{n'_j}{2}} \tag{34}$$

They represent the probability that a pair of points which are in the same cluster under C (respectively C') are also in the same cluster under the other clustering. In the context of Information Retrieval, W_I can be considered as *precision* and $\mathcal{W}_{II}$ can be considered as the *recall*.

However, $\mathcal{W}_I, \mathcal{W}_{II}$ are asymmetric measures.

B.3.3 Rand Index

B.3.3.1 General Rand Index

In standard classification problems, the result of a classification scheme is compared to a correct classification. The most common performance measure for this problem calculates the fraction of correctly classified (respectively misclassified) points to all points. Rand's Index (Rand, 1971) is the corresponding extension of the performance measure to the problem of comparing clusterings: *instead of counting single elements, we count correctly classified pairs of elements.* The Rand Index is defined as

$$\mathcal{R}(C, C') = \frac{N_{11} + N_{00}}{\binom{n}{2}} = \frac{\binom{n}{2} - \left[\sum_i \binom{n_i}{2} + \sum_j \binom{n'_j}{2}\right] + 2\binom{n_{ij}}{2}}{\binom{n}{2}} \tag{35}$$

$\mathcal{R}$ ranges from 0 (*no pair classified in the same way* under both clusterings) to 1 (*identical* clusterings). The value of R depends on both, the number of clusters and the number of elements.

The expected value of $\mathcal{R}$ can be computed as

$$E[\mathcal{R}] = 1 - \frac{\sum_i \binom{n_i}{2} + \sum_j \binom{n'_j}{2}}{\binom{n}{2}} + \frac{\left[\sum_i \binom{n_i}{2}\right]\left[\sum_j \binom{n'_j}{2}\right]}{\binom{n}{2}^2}$$

A problem with the Rand index is that the expected value of the Rand index of two random partitions does not take a constant value.

(Fowlkes & Mallows, 1983) show that in the (unrealistic) case of independent clusterings, $E[\mathcal{R}] \to 1$ (relatively quickly) as $K, K' \to n$, which is undesirable for a distance measure. They also show, $Var[\mathcal{R}] \to 1$ as $K, K' \to n$ and in practice $\mathcal{R}$ does not range over the entire $[0, 1]$ interval (*i.e.* $\min \mathcal{R} > 0$) and concentrates in a small interval near 1. Rand Index is also known to be highly dependent upon the number of clusters K (Morey & Agresti, 1984). (Discussed in Section B.3.5.1 below.)

B.3.3.2 Adjusted Rand Index

The expected value of the Rand Index of two random partitions does not take a constant value (e.g. zero). Thus, (Hubert & Arabie, 1985) proposed an adjustment where the two

clusterings are drawn randomly with a fixed number of clusters and a fixed number of points in each cluster (the number of clusters in the two clusterings need not be the same).

The adjusted Rand Index is the (normalized) difference of the Rand Index and it's expected value. It is defined as

$$\mathcal{AR}(C, C') = \frac{R(C, C') - E[R]}{1 - E[R]} = \frac{\binom{n_{ij}}{2} - \left[\sum_i \binom{n_i}{2}\right]\left[\sum_j \binom{n'_j}{2}\right] / \binom{n}{2}}{\left[\sum_i \binom{n_i}{2} + \sum_j \binom{n'_j}{2}\right] / 2 - \left[\sum_i \binom{n_i}{2}\right]\left[\sum_j \binom{n'_j}{2}\right] / \binom{n}{2}} \tag{36}$$

This index has expected value of 0 for *independent* clusterings and maximum value 1 (for *identical* clusterings).

However, it assumes independence between clusterings which is not valid in practice. Also, some pairs of clusterings may result in negative index values, which is absurd.

B.3.4 Fowlkes Mallows Index

(Fowlkes & Mallows, 1983) introduced a criterion for measuring hierarchical clusterings based on the Wallace measure which is symmetric. Though it was initially meant for comparing hierarchical clusterings, it can be used with equal ease for comparing flat clusterings as well. It is simply the geometric mean of the two Wallace measures $\mathcal{W}_I, \mathcal{W}_{II}$.

$$\mathcal{F}(C, C') = \sqrt{\mathcal{W}_I(C, C')\mathcal{W}_{II}(C, C')} = \frac{\sum_{i,j} \binom{n_{ij}}{2}}{\sqrt{(N_{11} + N_{10})(N_{11} + N_{01})}} = \frac{\sum_{i,j} \binom{n_{ij}}{2}}{\sqrt{(\sum_i \binom{n_i}{2})(\sum_j \binom{n'_j}{2})}} \tag{37}$$

In the context of Information Retrieval, this measure can be interpreted as the geometric mean of *precision* and *recall*. The main insight from $\mathcal{F}(C, C')$ was that the distance measure between two clusterings is not a one-dimensional concept. (Fowlkes & Mallows, 1983) give a detailed examples and show how to use the measure for selecting appropriate number of clusters and how two hierarchies can exhibit different degrees of similarity at different levels of cut.

Like for the adjusted Rand Index, the *measure* of similarity of two clusterings corresponds to the deviation from the expected value under the null hypothesis of independent clusterings with fixed cluster sizes. As with $\mathcal{AR}$, it also assumes independence between

clusterings which is not valid in practice. The index is used by subtracting the base-line and normalizing by the range, so that the expected value of the normalized index is 0 while the maximum (attained for *identical* clusterings) is 1. Some pairs of clusterings may theoretically result in negative indices under this normalization. For small numbers of clusters, the value of the index is very high, even for independent clusterings (which even achieve the maximum value for small numbers of clusters). This is highly undesirable.

B.3.5 Mirkin Metric

The Mirkin Metric which is also known as *Equivalence Mismatch Distance* is defined by

$$\mathcal{M}(C, C') = 2(N_{01} + N_{10}) = 2\binom{n}{2}[1 - \mathcal{R}(C, C')] = \sum_i \binom{n_i}{2} + \sum_j \binom{n'_j}{2} - \binom{n_{ij}}{2} \tag{38}$$

Thus, the Mirkin Metric is a variation of the Rand Index.

It is 0 for *identical* clusterings and positive otherwise. It corresponds to the Hamming distance for binary vectors if the set of all pairs of elements is enumerated and a clustering is represented by a binary vector defined on this enumeration. An advantage is the fact that this distance is a metric on P(X).

However, this measure is very sensitive to cluster sizes such that two clusterings that are *at right angles* (each cluster in one clustering contains the same amount of elements of each of the clusters of the other clustering) to each other are closer to each other than two clusterings for which one is a refinement of the other.

B.3.5.1 n-invariant Mirkin Metric

The n-invariant Mirkin Metric is defined as

$$\mathcal{M}_{inv}(C, C') = \frac{\mathcal{M}(C, C')}{n^2} \tag{39}$$

We get that

$$\mathcal{M}_{inv}(\hat{1}, C_K^U) = 1 - \frac{1}{K}$$

and

$$\mathcal{M}_{inv}(C, C') = \sum_{i=1}^{K} p_i^2 d(\hat{1}_{n_k}, C'_i) \tag{40}$$

whenever C' is a refinement of C and C'_i represents the partitioning induced by C' on cluster C_i of C.

Thus, The n-invariant Mirkin Metric satisfies the axioms $\mathcal{A}1 - \mathcal{A}3, \mathcal{A}4.M$ and $\mathcal{A}5.D$.

Thus, the invariant Mirkin metric is also aligned with the lattice of partitions. The *unnatural* behavior of the Rand index with increasing K has been noted earlier on and is the main reason why this index is used mostly in it's adjusted form.

B.3.6 Jaccard Index

The Jaccard Index is very common in geology and ecology, e.g. for measuring the species diversity between two different communities. It is very similar to the Rand Index, however it disregards the pairs of elements that are in different clusters for both clusterings. It is defined as follows:

$$\mathcal{J}(C, C') = \frac{N_{11}}{N_{11} + N_{10} + N_{01}} \tag{41}$$

B.3.7 Remarks

The measures presented in this section can all be calculated by means of the confusion matrix M (and the cluster sizes); this is either obvious from the formula (e.g. for the Fowlkes-Mallow Index) or can be seen after some transformation (e.g. for the Rand Index, which can be transformed into a variation of the Mirkin Metric).

For different reasons, these measures do not seem to be very appealing. Some of them are sensitive to certain parameters (cluster sizes, number of clusters, Figures 45, 46) ; think of a pair of clusterings with similarity $\alpha \in [0, 1]$ and replace each element in the underlying set by two elements. Why should the resulting pair of clusterings have a similarity other than α? This behavior, as well as sensitivity to the number of clusters, are undesirable.

Other measures, like the Fowlkes-Mallows Index, suffer from another drawback: they

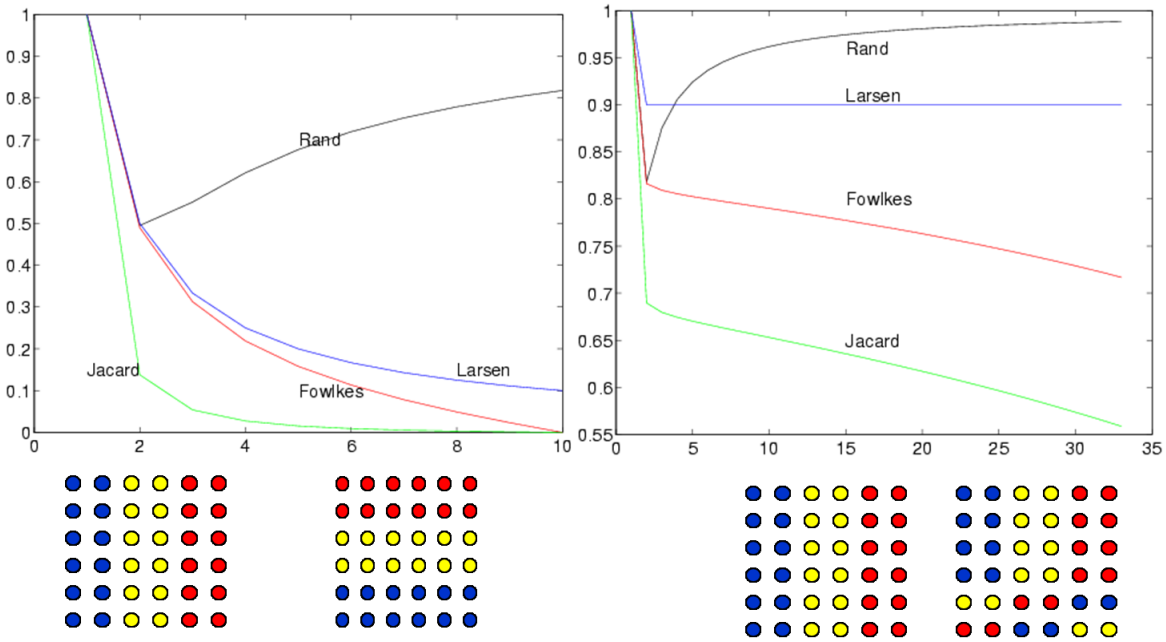


Figure 45: Sensitivity to Number of Clusters K

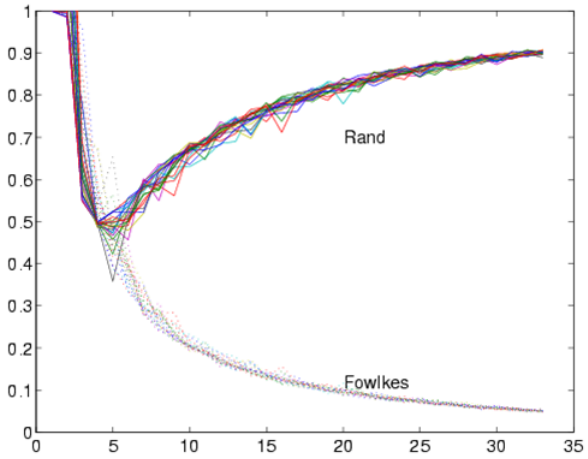


Figure 46: Variation of the baseline for different clusterings with the same K

make use of a very strong null hypothesis, that is, independence of the clusterings, fixed number of clusters, and fixed cluster sizes. When comparing results provided by clustering algorithms these assumptions are - apart from the number of clusters that is fixed for some algorithms - violated. None of the algorithms works with fixed cluster sizes. Furthermore, in practice it would be against the intuition to compare two clusterings when assuming that there is no relationship between them. In fact, we compare clusterings because we suppose

a certain relationship and we want to know how strong it is.

B.4 Measures based on Sets

A second category of criteria is based on set cardinality alone and does not make any assumption about how the clusterings may have been generated. These measures try to match clusters that have a maximum absolute or relative overlap. This is also a quite intuitional approach, however, the asymmetry of some of the measures makes them difficult to use.

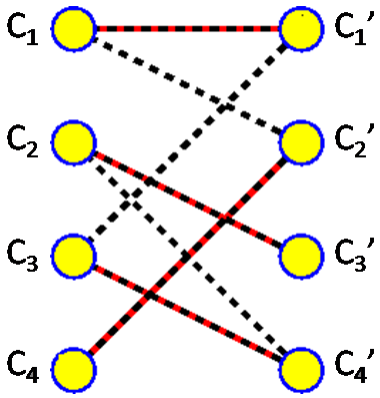


Figure 47: Set Matching

B.4.1 $\mathcal{F}$-Measure

n statistics, $\mathcal{F}$-Measure is a measure of a test's accuracy. It considers both the *precision* and *recall* of the test to compute the score: *precision* is the number of correct results divided by the number of all returned results and *recall* is the number of correct results divided by the number of results that should have been returned. $\mathcal{F}$-Measure can be interpreted as a weighted average of the precision and recall, where it reaches it's best value at 1 and worst at 0. It is used in the field of information retrieval for measuring search, document classification, and query classification performance.

It was first used in the field of document clustering where it is used to evaluate the accuracy of a clustering solution. Each cluster of the first clustering is a (predefined) *class*

of documents and each cluster of the second clustering is treated as the *result of a query*.

The $\mathcal{F}$-Measure for a *cluster* C'_j with respect to a certain *class* C_i indicates how *good* the cluster C'_j describes the class C_i by calculating the harmonic mean of precision and recall, where precision is defined as

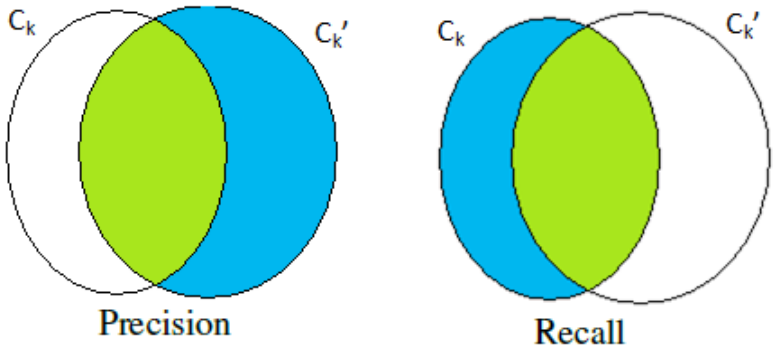


Figure 48: Precision and Recall

$$\mathcal{P}(C_i, C'_j) = \frac{n_{ij}}{n'_j} \tag{42}$$

and recall is defined as

$$\mathcal{R}(C_i, C'_j) = \frac{n_{ij}}{n_i} \tag{43}$$

Thus $\mathcal{F}$-Measure is defined as

$$\mathcal{F}(C_i, C'_j) = \frac{2\mathcal{P}(C_i, C'_j)\mathcal{R}(C_i, C'_j)}{\mathcal{P}(C_i, C'_j) + \mathcal{R}(C_i, C'_j)} = \frac{n_{ij}}{n_i + n'_j} \tag{44}$$

The overall $\mathcal{F}$-Measure is then defined as the weighted sum of the maximum $\mathcal{F}$-Measures for the clusters in C'

$$\mathcal{F}(C, C') = \mathcal{F}(C') = \frac{1}{n} \sum_{i=1} K n_i \max_{j=1}^{K'} \mathcal{F}(C_i, C'_j) \tag{45}$$

This measure is not symmetric. Thus, it may be appropriate only for comparing a clustering with an optimal clustering solution. However, in general the optimal solution is not known, which makes an asymmetric measure hard to interpret.

B.4.2 Meilă and Heckerman Measure

This measure is also known as the *Classification Error*. First, each cluster of C is given a *best match* in C'. Then, $\mathcal{H}$ is computed as the total *unmatched probability mass* in the confusion matrix M. This is defined as

$$\mathcal{H}(C, C') = 1 - \frac{1}{n} \max_{\pi} \sum_{i} n_{i,\pi(i)} \tag{46}$$

In the above, π is an injective mapping of $\{1, 2, \ldots, K\}$ into $\{1, 2, \ldots, K'\}$ and the maximum is taken over all such mappings.

In other words, for each π we have a (partial) correspondence between the cluster labels in C and C'; now looking at clustering as a classification task with the fixed label correspondence, we compute the *classification error* of C' w.r.t. C. The minimum possible classification error under all correspondences is $\mathcal{H}$.

We have that

$$\mathcal{H}(\hat{1}, C_K^U) = 1 - \frac{1}{K}$$

Thus, Meilă and Heckerman Measure satisfies axioms $\mathcal{A}1$, $\mathcal{A}4$, $\mathcal{A}5.D$. But it violates axioms $\mathcal{A}2$ and $\mathcal{A}3$. Thus, it is not aligned to the lattice of partitions.

$\mathcal{H}$ can be computed in polynomial time by *Max bi-partite matching (Hungarian) algorithm* on confusion matrix M. It takes value 1 for identical clusterings.

B.4.3 Van Dongen-Measure

Van Dongen introduced a symmetric measure, that is also based on maximum intersections of clusters. It is defined as follows:

$$\mathcal{V}(C, C') = 2n - \sum_{i} \max_{j} n_{ij} - \sum_{j} \max_{i} n_{ij} \tag{47}$$

This measure is a metric on the space of all clusterings of the underlying set D. However, it ignores the parts of the clusters outside the intersections.

B.4.3.1 n-invariant van Dongen Metric

$$\mathcal{V}_{inv}(C, C') = \frac{\mathcal{V}(C, C')}{2n} \tag{48}$$

We get that

$$d(\hat{1}, C) = \frac{1}{2}\left[1 - \frac{\max_k n_k}{n}\right] \tag{49}$$

and

$$\mathcal{V}_{inv}(\hat{1}, C_K^U) = \frac{1}{2}\left(1 - \frac{1}{K}\right)$$

Thus, The n-invariant van Dongen Metric satisfies the axioms $\mathcal{A}1, \mathcal{A}2D, \mathcal{A}3, \mathcal{A}4 and \mathcal{A}5.D$. Thus, the van Dongen metric is horizontally aligned with the lattice of partitions but not vertically.

B.4.4 Remarks

These measures have the common property of just taking the overlaps into account. Thus, they all suffer from the *problem of matching*. They completely disregard the unmatched parts of the clusters (or even complete clusters).

This is illustrated in the discussion below: One way or another, they all first find a *best match* for each cluster, then add up the contributions of the matches found. In doing so, the criteria completely ignore what happens to the *unmatched* part of each cluster.

To make things clear, let us look at the example depicted in Figure 49. Suppose C is a

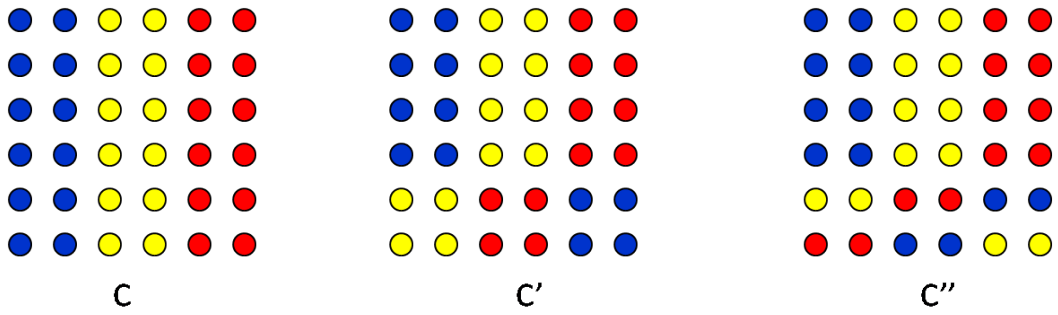


Figure 49: Ignored Unmatched Parts in Set Matching

clustering with K equal clusters. The clustering C'' is obtained from C by moving a fraction

f of the points in each C_i to the cluster $C_{(i+1) mod\ K}$. The clustering C' is obtained from C by reassigning a fraction f of the points in each C_i evenly between the other clusters. If f ¡ 0.5 then $\mathcal{F}(C, C') = \mathcal{F}(C, C'')$, $\mathcal{H}(C, C') = \mathcal{H}(C, C'')$ and $D(C, C') = D(C, C'')$. This contradicts the intuition that C' is a less disrupted version of C than C''.

These methods are good if the clusterings are very close together. However, they perform extremely badly in case the clusterings are very different.

B.5 Measures based on Mutual Information

This approach to the comparison of clusterings is based on the notion of *entropy* and has it's origin in *Information Theory*. When applied to clusterings, the meaning of entropy can be described as follows: Assume that all points in D have the same probability of being picked and choosing a point in X at random, the probability that this point is in cluster $C_i \in C$ is $p_i = \frac{n_i}{n}$. Thus, entropy associated with a clustering can be defined as

$$\mathcal{H}(C) = -\sum_i p_i \log p_i \tag{50}$$

This can computed from the normalized Confusion Matrix P.

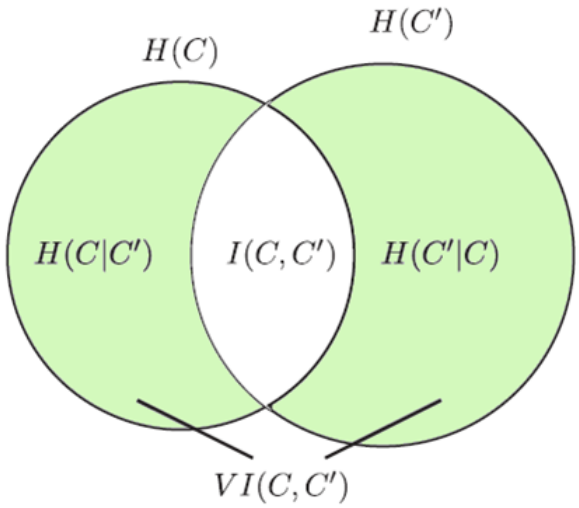


Figure 50: Mutual Information

The entropy of a clustering C is a measure for the uncertainty about the cluster of a

randomly picked point. In the case of a trivial clustering (1 cluster or n clusters), we know the cluster of a randomly picked point, thus the entropy of such a clustering is 0.

The notion of entropy can be extended to that of *mutual information*, which describes how much we can on an average reduce the uncertainty about the cluster of a random element when knowing it's cluster in another clustering of the same set of points. Formally, the mutual information between two clusterings C, C' is defined as

$$\mathcal{I}(C, C') = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{p_i p_j} \tag{51}$$

This can again be computed from the normalized Confusion Matrix P. The mutual information $\mathcal{I}$ is a metric on the space of all clusterings. However, it is not bounded by a constant value which makes it difficult to interpret.

Like the Rand index, the base line value of mutual information between two random clusterings does not take on a constant value, and tends to be larger when the two clusterings have a larger number of clusters.

B.5.1 Normalized Mutual Information

Strehl and Ghosh introduce the concept of combining multiple clusterings into a single one without accessing the original features or algorithms that determined these clusterings. For this purpose, they (approximately) determine the clustering that has the maximal average normalized mutual information with all the clusterings in consideration, where they defined normalized mutual information between two clusterings as

$$\mathcal{NMI}_1(C, C') = \frac{\mathcal{I}(C, C')}{\sqrt{H(C)H(C')}} \tag{52}$$

In order to obtain a good and robust clustering of a given set of points, Fred and Jain propose to combine the results of multiple clusterings instead of using just one particular algorithm. They used an optimization criterion searching for the clustering that maximizes the average normalized mutual information with all the clusterings, where they defined

normalized mutual information between two clusterings as

$$\mathcal{NMI}_2(C, C') = \frac{2\mathcal{I}(C, C')}{H(C) + H(C')} \tag{53}$$

Both measures lie in the range $[0, 1]$ and take the value 1 for identical clusterings and 0 for clustering that have no points in common or in the degenerate case when $\frac{p_{ij}}{p_i p_j} = 1$ for all i, j pairs. This happens when one of the clusterings is the trivial.

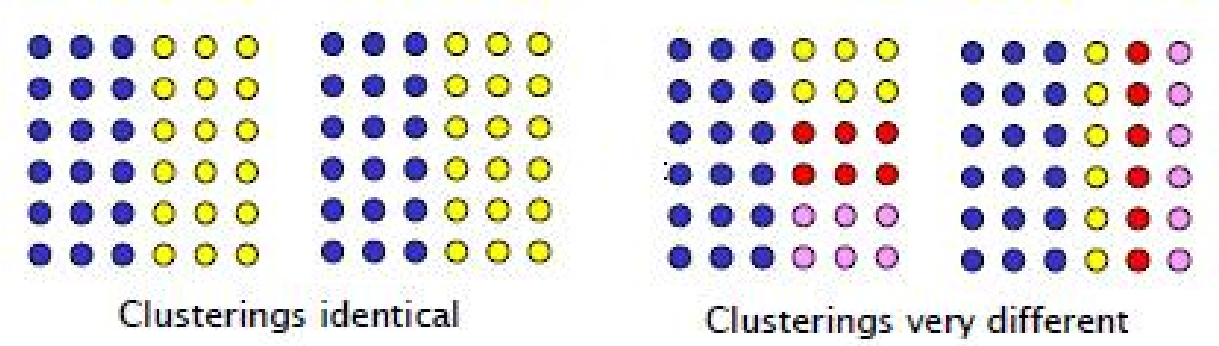


Figure 51: Mutual Information is not enough. For both cases I(C, C') = 1

However, Mutual Information is not enough. Figure 51 shows that even though two clusterings could differ by a lot, still the measures could interpret them as equivalent.

B.5.2 Variation of Information

This measure was proposed by Meilă (Meila, 2003, 2002)and is also based on entropy. It is defined as

$$\mathcal{VI}(C, C') = \mathcal{H}(C) + \mathcal{H}(C') - 2\mathcal{I}(C, C') = \mathcal{H}(C|C') + \mathcal{H}(C'|C) \tag{54}$$

The first term corresponds to the amount of information about C that we lose, while the second term corresponds to the amount of information about C' that we gain, when going from clustering C to C'.

$\mathcal{VI}$ is the only information-theoretical measure that has a detailed analysis. It has been proven in (Meilă, 2005; Meilă, 2007) that it satisfies all the axioms $\mathcal{A}1 - \mathcal{A}5$. Other properties of $\mathcal{VI}$ are summarized below:

- $\mathcal{VI}$ is a metric.

- $\mathcal{VI}$ is a n-invariant. It depends only on the relative sizes of the clusters. It does not directly depend on the number of points in the data set

- $\mathcal{VI}$ is not bounded by a constant value. There is an upper bound of

$$\mathcal{VI}(C, C') \leq \log n$$

(which is attained for all n, e.g. with the two trivial clusterings) and if the number of clusters is bounded by a constant K, with $K \leq \sqrt{n}$, then $\mathcal{VI}(C, C') \leq 2 \log K$; This means that for large enough n, clusterings of different data sets, with different numbers of elements, but with bounded numbers of clusters are on the same scale in the metric $\mathcal{VI}$. This allows us to compare, add or subtract $\mathcal{VI}$ distances across different clustering spaces independently of the underlying data set.

- Local Neighborhood. Axiom $\mathcal{A}3$ implies $\mathcal{VI}(C, C') \geq \mathcal{VI}(C, C \times C')$. Thus, the nearest neighbor of a clustering C is either a refinement of C or a clustering of which C is a refinement. In fact, the nearest neighbor of a clustering is obtained by splitting one element off the smallest cluster (or by the corresponding merge process). This means that small changes in a clustering result in small $\mathcal{VI}$ distances.

- $\mathcal{VI}$ between two clusterings C, C' with $C \neq C'$ has a lower bound of $2/n$. Thus, with increasing n, the space of clusterings gets a finer granularity.

- $\mathcal{VI}$ can be computed in $O(n + KK')$ time. We need time $O(n)$ for computing the confusion matrix M and time $O(KK')$ for computing $VI(C, C')$ from M.

B.5.3 Remarks

The information theory based measures seem to be quite promising because they overcome the drawbacks with measures based on counting pairs or on set overlaps. However,

they possibly suffer from other disadvantages that have not been realized yet. An extensive examination of these measures, especially the two versions of the normalized mutual information, is necessary.

B.6 An Impossibility Result

There is no symmetric, n-invariant, distance measure d having non-decreasing $d(\hat{1}; C_K^U)$ that satisfies simultaneously the following three properties:

- d is aligned to the lattice of partitions (axioms $\mathcal{A}2, \mathcal{A}3$).
- d is Convex Additive (axiom $\mathcal{A}4$).
- d is bounded.

The *first* property gives geometric intuition in addition to the intuition one would get from having a metric. Preserving this property would be useful in designing search algorithms in the space of clusterings and proving their properties.

The *second* one is the most important for the *understandability* of a distance, because it is a law of composition. It says that under unions of sets, the distances on the parts are weighted in proportion to the sizes of the parts and then summed.

The *third* property is useful as a distance that is bounded (for instance between 0 and 1) can be interpreted as a probability. Moreover, in statistics, the tradition is to indicate identity between two clusterings by a 1.

B.7 Discussion

There are many competing criteria for comparing clusterings, with no clear best choice. It is as meaningless to search for a best criterion for comparing clusterings, just as it is to search for the best clustering algorithm. There is no criterion for comparing clusterings that fits every problem optimally. Algorithms are good in as much as they match the task

at hand. If a clustering algorithm performs better than other clustering algorithms on many of these measures, we can have some confidence in the better algorithm.

With respect to distances between clusterings, the goal is to better understand their properties, their limitations, and the implied assumptions underlying them so that a user of distances between clusterings can make an informed decision while making a choice. Characterizing distances between clusterings in terms of axioms highlights the essential properties of these distances, from which all others follow.

Clustering is a hard problem and seeing clusterings as nodes in a graph opens the possibility of applying new techniques, based on graph and lattice theory, to this task.

APPENDIX C

CONCENTRATION BOUNDS

Let X be a random variable and let $x \in R$. Then $Pr[X > x]$ denotes the upper tail and $Pr[X < x]$ denotes the lower tails. Figure 52 illustrates the tails. Concentration bounds give upper bounds for the tails of the random variable $X - E[X]$.

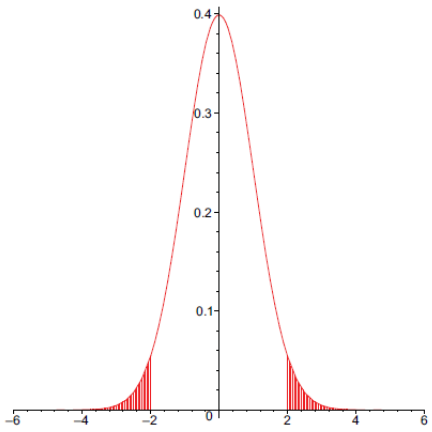


Figure 52: Tails of a Random Variable

C.1 Properties of Random Variables

We define certain properties of the random variables that are used frequently during the proofs. For any random variables X, Y and any constant $c \in R$.

$$
\begin{aligned}
E[X+Y] &= E[X] + E[Y] \\
E[cX] &= cE[X] \\
Var[X] &= E[(X - E[X])^2] \\
Var[cX] &= c^2 Var[X]
\end{aligned}
$$

If X, Y are independent

$$\begin{aligned} E[XY] &= E[X]E[Y] \\ Var[X+Y] &= Var[X] + Var[Y] \end{aligned}$$

Lemma 13. *Let $t > 0$ and X be a random variable. Then for any integer $k > 0$*

$$Pr\left[|X| \geq t\right] \leq \frac{E[|X|^k]}{t^k} \tag{55}$$

Proof:

$$\begin{aligned} E[|X^k|] &= \sum_x |x|^k Pr[X = x] \\ &= \sum_{|x| \leq t} |x|^k Pr[X = x] + \sum_{|x| > t} |x|^k Pr[X = x] \\ &\geq \sum_{|x| > t} |x|^k Pr[X = x] \\ &\geq t^k \sum_{|x| > t} Pr[X = x] \\ &= t^k Pr[|X| > t] \end{aligned}$$

■

C.2 Markov Inequality

This gives an upper bound for the probability that a *non-negative* function of a random variable is greater than or equal to some positive constant. This is also known as the first moment method.

Theorem 19. *Let X be a* positive *random variable. Then, for any $t > 0$*

$$Pr\left[X \geq t\right] \leq \frac{E[X]}{t} \tag{56}$$

Proof: From Lemma 13, we know $Pr\left[|X| \geq t\right] \leq \frac{E[|X|^k]}{t^k}$. Setting $k = 1$, we get the desired result. ■

Corollary 3. *Let $X_1, X_2, ...X_n$ be independent random variables. Let X be the random variable defined by $X = \sum_{i=1}^{n} X_i$. Then, for any $c > 0$ and any $t > 0$*

$$Pr\left[X > t\right] \leq e^{-ct} \prod_{i=1}^{n} E\left[e^{cX_i}\right] \tag{57}$$

Proof:

We consider a random variable e^{cX} instead of X. For $c > 0, e^{cX}$ is a positive monotone increasing function of X and therefore

$$Pr\left[X > t\right] = Pr[e^{cX} > e^{ct}]$$

Using Markov's inequality we get

$$\begin{aligned} Pr\left[X > t\right] &= Pr[e^{cX} > e^{ct}] \leq \frac{E[e^{cX}]}{e^{ct}} \\ &= \frac{E\left[e^{c\sum_{i=1}^{n} X_i}\right]}{e^{ct}} \\ &= \frac{E\left[\prod_{i=1}^{n} e^{cX_i}\right]}{e^{ct}} \\ &= e^{-ct} \prod_{i=1}^{n} E\left[e^{cX_i}\right] \qquad \text{by linearity of expectation} \end{aligned}$$

■

C.2.1 Example: Coin Toss

From Markov inequality we get,

$$Pr\left[X \geq \mu + t\right] \leq \frac{\mu}{\mu + t} = \frac{1}{1 + 2t/n}$$

C.2.2 Remarks

Markov inequality relates probabilities to expectations, and provides a loose bound for the cumulative distribution function of a random variable. Markov inequality does not provide any information on the lower tail. It is useless in range $0 < t < 1$. It cannot be used with the random variable $-X$. Markov inequality does not depend on any knowledge of the distribution of X. It is not a tight bound.

C.3 Chernoff Bounds

First, we present a useful lemma.

Lemma 14. *Let $X_1, X_2, ...X_n$ be a sequence of independent Bernoulli random variables, each variable having a probability p_i of success and probability $q_i = 1 - p_i$ of failure. Let X be the random variable defined by $X = \sum_{i=1}^{n} X_i$. Then, for any $c > 0$ and any $t > 0$*

$$Pr\left[X > t\right] \leq e^{-ct} \prod_{i=1}^{n} (p_i e^c + q_i) \tag{58}$$

and

$$Pr\left[X < t\right] \leq e^{ct} \prod_{i=1}^{n} (p_i e^{-c} + q_i) \tag{59}$$

Proof: For each $i \in [n]$, $e^{cX_i} = e^c$ with probability p_i and $e^{cX_i} = e^c$ with probability q_i. Thus, $E[e^{cX_i}] = p_i e^c + q_i$. Plugging in the value in Eq. (57) of Corollary 3, we get

$$Pr\left[X > t\right] \leq e^{-ct} \prod_{i=1}^{n} (p_i e^c + q_i)$$

This gives the upper tail.

Similarly we can get the proof for the lower tail. ■

Theorem 20. *Let $X_1, X_2, ...X_n$ be a sequence of independent Bernoulli random variables, each variable having a probability p_i of success and probability $q_i = 1 - p_i$ of failure. Let X be the random variable defined by $X = \sum_{i=1}^{n} X_i$ with expected value $\mu = \sum_{i=1} np_i$. Then, for any $d > 0$*

$$Pr\left[X > (1+d)\mu\right] \leq \left(\frac{e^d}{(1+d)^{1+d}}\right)^{\mu} \tag{60}$$

For the case $0 < d < 1$, we can also define the lower tail as

$$Pr\left[X < (1-d)\mu\right] \leq \left(\frac{e^{-d}}{(1-d)^{1-d}}\right)^{\mu} \tag{61}$$

Proof: We first give a proof of the upper tail. Let $t = (1+d)\mu$ and $c = \ln(1+d)$. Using in Eq. (65) of Lemma 14, we get

$$\begin{aligned} Pr\left[X > t\right] &\leq e^{-(1+d)\mu \ln(1+d)} \prod_{i=1}^{n} (p_i e^{\ln(1+d)} + q_i) \\ &= (1+d)^{-(1+d)\mu} \prod_{i=1}^{n} (p_i(1+d) + (1-p_i)) \\ &= \frac{\prod_{i=1}^{n}(1+dp_i)}{(1+d)^{(1+d)\mu}} \\ &\leq \frac{\prod_{i=1}^{n} e^{dp_i}}{(1+d)^{(1+d)\mu}} \\ &\leq \left(\frac{e^d}{(1+d)^{1+d}}\right)^{\mu} \end{aligned}$$

This gives the upper tail.

For $0 < d < 1$, we take $t = (1-d)\mu$ and $c = -\ln(1-d)$. Using in Eq. (59) of Lemma 14, and using a similar proof as above, we get the lower bound. ■

Corollary 4. *Let $X_1, X_2, ...X_n$ be a sequence of independent Bernoulli random variables, each variable having a probability p_i of success and probability $q_i = 1 - p_i$ of failure. Let X be the random variable defined by $X = \sum_{i=1}^{n} X_i$ with expected value $\mu = \sum_{i=1} np_i$. Then, for any $0 < d < 1$*

$$Pr\left[X > (1+d)\mu\right] \leq exp\left(-\frac{d^2\mu}{3}\right) \tag{62}$$

and

$$Pr\left[X < (1-d)\mu\right] \leq exp\left(-\frac{d^2\mu}{2}\right) \tag{63}$$

Therefore,

$$Pr\left[|X - \mu| > d\mu\right] \leq 2exp\left(-\frac{d^2\mu}{3}\right) \tag{64}$$

Proof: Using Taylor's expansion around 0 we have

$$(1-d)\ln(1-d) = -d + \frac{d^2}{2} + O(d^3)$$

This implies,

$$(1-d)^{1-d} > e^{\frac{d^2}{2-d}}$$

Substituting this in Eq. (61) we get

$$\begin{aligned} Pr\left[X < (1-d)\mu\right] &\leq \left(\frac{e^{-d}}{e^{\frac{d^2}{2-d}}}\right)^{\mu} \\ &\leq exp\left(-\frac{d^2\mu}{2}\right) \end{aligned}$$

To get the upper tail, we claim that for $0 < d < 1$ we have $d - (1+d)\ln(1+d) + \frac{d^2}{3} \leq 0$. (It is true for $d = 0, d = 1$. To prove it for $d \in (0,1)$, let $f(d) = d - (1+d)\ln(1+d) + \frac{d^2}{3}$. Taking the first derivative, $f'(d) = -\ln(1+d) + \frac{3d}{2}$. The roots are $d = 0$ and $d = 1.144$. Since f is continuous and has no root in the range $(0,1)$ it is monotonic in the range. Since $f(0)$ and $f(1)$ both are ≤ 0, the function is monotonically decreasing. Thus, we get

$$(1+d)\ln(1+d) \geq d + \frac{d^2}{3}$$

Thus,

$$(1-d)^{1-d} \geq e^{d+\frac{d^2}{3}}$$

Substituting this in Eq. (60) we get

$$\begin{aligned} Pr\left[X > (1+d)\mu\right] &\leq \left(\frac{e^{d}}{e^{d+\frac{d^2}{3}}}\right)^{\mu} \\ &\leq exp\left(-\frac{d^2\mu}{3}\right) \end{aligned}$$

A union bound on Eq. (62) and Eq. (63) gives Eq. (64). ■

C.3.1 The Hoeffding Extension

An extension by the same basic technique is possible to variables that need not even be discrete. To calculate the moment-generating function e^{cX}, we need, as before, to compute each individual e^{cX_i} . This is no longer as simple as it was with the case where X_i took only two values. However, the following Lemma gives a simple upper bound.

Lemma 15. *Let $X_1, X_2, ...X_n$ be a sequence of independent random variables, each variable X_i takes values in the range $[0, 1]$ and has a mean p_i. Let X be the random variable defined by $X = \sum_{i=1}^{n} X_i$. Then, for any $c > 0$ and any $t > 0$*

$$Pr[X > t] \leq e^{-ct} \prod_{i=1}^{n} (p_i e^c + q_i) \tag{65}$$

Proof: The graph of the function e^{cX} is convex and hence, in the interval $[0, 1]$, lies always below the straight line joining the endpoints $(0, 1)$ and $(1, e^c)$. This line has the equation $y = ax + b$ where $b = 1$ and $a = e^c - 1$. Thus, $E[e^{cX_i}] \leq E[aX_i + b] = p_i e^c + q_i$. Plugging in the value in Eq. (57) of Corollary 3, we get

$$Pr[X > t] \leq e^{-ct} \prod_{i=1}^{n} (p_i e^c + q_i)$$

This gives the desired result. ■

The Lemma 15 can be used accordingly in the proof of Theorem to get a similar result for continuous random variables.

C.3.2 Example: Coin Toss

Let us consider the problem of counting the number of heads when tossing n times a fair coin. Let $X = (X_1, X_2....X_n)$ be a random vector where for each $i \in [n]$ the probability that the i_{th} toss is a head is $Pr[X_i = 1] = \frac{1}{2}$. The expected value for X_i, $E[X_i] = 1/2$ and the expected number of heads is $E[X] = n/2$. The variance $Var[X]\sigma^2 = n/4$.

For $t = d\mu$, we get

$$Pr[X - \mu > t] \leq exp\left(-\frac{t^2}{3\mu}\right) = exp\left(-\frac{2t^2}{3n}\right)$$

C.3.3 Remarks

Chernoff bounds are especially convenient for bounding tail probabilities of sum of independent random variables. This is because the expected value of product of independent variables equals the product of the expected value (Lemma 14)and the problem of finding

tight bounds reduces to finding good upper bounds for the moment generating functions of the random variables $X_i - E[X_i]$. Chernoff bounds have very useful applications in set balancing, Load balancing, probabilistic amplification, packet routing in sparse networks, etc. Chernoff bounds are also used to obtain tight bounds for permutation routing problems which reduce network congestion while routing packets in sparse networks.

APPENDIX D

MATLAB IMPLEMENTATION

We briefly describe below the MATLAB implementation done for the experimental analysis (Detailed help documents for each class are available with the code). Different MATLAB classes have been developed as necessary to successfully implement clustering algorithms. These classes can be grouped by purpose such as *data sets, Similarity Information, Hierarchy, Standard Linkage Algorithms, General Hierarchical Algorithms, Partitional Algorithms* and *Output*.

D.1 Data Sets

D.1.1 Class *abstdataset*

For loading and manipulation various machine learning datasets a base class *abstdataset* has been implemented, which provides all the basic operations like similarity, error calculation, graph generation, etc. required for any data set. This class is abstract and cannot be used directly

D.1.2 Class *obsvdataset*

For simple data sets with regular feature sets and label information, the class *obsvdataset* has been implemented through which different datasets can be loaded with ease. The only requirement for this class is that the feature set contain the label information under variable *class*. As an example, data set *iris* has been added at location *"data sets/iris/iris.data"* for reference.

Example Usage:

The data set can be loaded as:

```
ds = obsvdataset('data sets/iris/iris.data');
```

Note: For data sets with special requirements they can be directly implemented deriving the *abstdataset* class. *e.g.* *digits*, the digits data set has every observation as a digit image of dimension 28x28. To deal with such tricky data sets, specific implementations are preferred.

D.1.3 Class *inductivedataset*

For data sets using an inductive setting, in which the core algorithm is run only on a randomly chosen subset of points in the actual data set, the class *inductivedataset* has been implemented. An inductive data set can be generated specifying the fraction of points to keep.

Example Usage:

To create an inductive data set of *iris* using a sample of 0.2 fraction of the points.

```
ds = inductivedataset('data sets/iris/iris.data', 0.2);
```

D.2 Similarity Information

D.2.1 Class *similarity*

To generate similarity information for a given data set the base class *similarity* has been implemented which contains the basic information regarding similarity information.

Similarity information can be generated using different similarity functions, *e.g.* *euclidean*, *cosine*, etc. Various merge methods such as *single_linkage*, *complete_linkage*, etc. have also been implemented.

However, the user is free to manipulate the similarity matrix on their own and not use any of these standard merging methods.

Usually this class is linked to a particular data set, *e.g. obsvSimilarity, digitsSimilarity* and is initialized inside the dataset class. However, users are free to create their own similarity for any data set and associate it with a data set.

D.3 Hierarchy

D.3.1 Class *CLTreeNode*

This class implements each node of the tree. It maintains the information regarding a nodes children, height, parents, clusters, etc.

D.3.2 Class *CLTree*

To maintain the hierarchy generated by the algorithms, class *CLTree* has been implemented. It maintains the cluster information after each merge operation. *CLTree* maintains the list of the roots of the forest. It also provides a default drawing function to draw the hierarchy.

Example Usage:

To create a cluster tree with one root node:

```
node = CLTreeNode([1 2 4 5]);
clTree = CLTree();
clTree.addRoot(node);
```

D.4 Hierarchical Algorithms

D.4.1 Class *agcl_base*

To implement different agglomerative clustering algorithms, the base class *agcl_base* has been implemented which contains the basic information required for any agglomerative algorithm.

D.4.2 Standard Linkage Algorithms

Class *aggclust*

The standard linkage algorithms are implemented using class *aggclust* which derives from *agcl_base*. This class has been wrapped in the *linking* class which does various pre-post processing steps for ease of use, such as computing errors of clusterings, generating required graphs after running the clustering algorithm, etc. In general, this class should be used rather than the algorithm class *agcl_base* directly.

Example Usage:

To run single linkage on a dataset do:

```
linking(ds, 'single_linkage')
```

D.4.3 Other Hierarchical Algorithms

Above holds for all algorithms in general, as for ease of use, all hierarchical algorithms have been wrapped in a class that does basic pre-post processing steps, *e.g.* if the algorithm is named *x*, *x_linking* would be the wrapping class (with minor variations).

Example Usage:

To run the *x_linking* class, simply do:

```
x_linking(ds);
```

x here refers to various algorithms such as:

bullet *fast_nuss*: The robust hierarchical clustering algorithm.

bullet *weighted_nbr*: The weighted neighborhood linkage algorithm.

bullet *interesting*: The interesting linkage algorithm.

bullet *ranked_tree*: The symmetric statistical linkage algorithm.

bullet and so on.

D.5 Partitional Algorithms

Several partitional algorithm classes have also been implemented.

D.5.1 Class *kmeans_cluster*

Class *kmeans_cluster* is a wrapper over *kmeans* that runs it multiple times for fairness and analyses the results.

D.5.2 Class *em_cluster*

Class *em_cluster* is similarly a wrapper over *em* algorithm.

D.5.3 Class *dbscan_cluster*

Class *em_cluster* is similarly a wrapper over *dbscan* algorithm.

D.6 Output

The hierarchies are output as tree, implemented in class *CLTree*. The clustering are output as scatter graphs, implemented in class *abstdataset*. The Classification Error is computed using the Hungarian method, implemented in class *abstdataset*.

REFERENCES

Aggarwal, C.C., Wolf, J.L., Yu, P.S., Procopiuc, C., and Park, J.S. Fast algorithms for projected clustering. *ACM SIGMOD Record*, 28(2):61–72, 1999. ISSN 0163-5808.

Anderberg, M R. Cluster analysis for applications, 1973.

Arabie, P., Hubert, L.J., and De Soete, G. *Clustering and classification*. World Scientific Pub Co Inc, 1996. ISBN 9810212879.

Balcan, M.-F. and Blum, A. On a theory of learning with similarity functions. In *International Conference on Machine Learning*, 2006.

Balcan, M.-F., Blum, A., and Vempala, S. A discrimantive framework for clustering via similarity functions. In *40th ACM Symposium on Theory of Computing*, 2008.

Balcan, Maria-Florina and Gupta, Pramod. Robust hierarchical clustering. In *Conference on Learning Theory*, 2010a.

Balcan, Maria-Florina and Gupta, Pramod. Weighted neighborhood linkage: A robust algorithm for hierarchical clustering. In *NIPS-ROBUSTML*, 2010b.

Bennett, K.P. *Decision tree construction via linear programming*. Center for Parallel Optimization, Computer Sciences Dep., Univ. of Wisconsin, 1992.

Blum, Manuel, Floyd, Robert W., Pratt, Vaughan R., Rivest, Ronald L., and Tarjan, Robert Endre. Linear time bounds for median computations. In *STOC*, pp. 119–124, 1972.

Bobisud, HM and Bobisud, LE. A metric for classifications. *Taxon*, 21(5):607–613, 1972. ISSN 0040-0262.

Chaudhuri, Kamalika and Dasgupta, Sanjoy. Rates of convergence for the cluster tree. In Lafferty, J., Williams, C. K. I., Shawe-Taylor, J., Zemel, R.S., and Culotta, A. (eds.), *Advances in Neural Information Processing Systems 23*, pp. 343–351. 2010.

Cheng, D., Kannan, R., Vempala, S., and Wang, G. A divide-and-merge methodology for clustering. *ACM Trans. Database Syst.*, 31(4):1499–1525, 2006.

Coppersmith, D. and Winograd, S. Matrix multiplication via arithmetic progressions. In *Proceedings of the 19-th annual ACM conference on Theory of computing*, pp. 1–6, 1987.

Cristianini, N., Shawe-Taylor, J., Elisseeff, Andre, and Kandola, J. On kernel target alignment. In *Advances in Neural Information Processing Systems*, 2001.

Dasgupta, A., Hopcroft, J. E., Kannan, R., and Mitra, P. P. Spectral clustering by recursive partitioning. In *ESA*, pp. 256–267, 2006.

Dasgupta, S. and Long, P. Performance guarantees for hierarchical clustering. *Journal of Computer and System Sciences*, 70(4):555 – 569, 2005.

Duda, R. O., Hart, P. E., and Stork, D. G. *Pattern Classification*. Willey, 2000.

Edwards, AWF and Cavalli-Sforza, LL. A method for cluster analysis. *Biometrics*, pp. 362–375, 1965. ISSN 0006-341X.

Everitt, BS. Cluster Analysis. 1993. *London: Edward Arnold*, 1993.

Fisher, R.A. et al. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7:179–188, 1936.

Florek, K., Lukaszewicz, J., Perkal, J., Steinhaus, H., and Zubrzycki, S. Sur la liaison et la division des points d'un ensemble fini. In *Colloquium Mathematicum*, volume 2, pp. 282–285, 1951.

Fowlkes, E B and Mallows, C L. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association 78(383):553 569. American Statistical Association*, 1983.

Frank, A. and Asuncion, A. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`. [Date Accessed: 29/03/2011]. University of California, Irvine, School of Information and Computer Science.

Frigui, Hichem and Krishnapuram, Raghu. Clustering by competitive agglomeration. *Pattern Recognition*, 30(7):1109–1119, 1997.

Gower, J. C. A comparison of some methods of cluster analysis. *Biometrics*, 23(4):623–637, 1967. ISSN 0006341X.

Gower, J.C. A general coefficient of similarity and some of its properties. *Biometrics*, pp. 857–871, 1971. ISSN 0006-341X.

Guha, S, Rastogi, R, and Shim, K. Rock: A robust clustering algorithm for categorical attributes. In *In Proc. of the 15th Intl Conf. on Data Eng*, 1999.

Guha, Sudipto, Rastogi, Rajeev, and Shim, Kyuseok. Cure: an efficient clustering algorithm for large databases. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pp. 73–84, New York, NY, USA, 1998. ACM. ISBN 0-89791-995-5.

Hartigan, J A and Wong, M A. A k-means clustering algorithm. *Applied Statistics*, (28): 100–108, 1979.

Herbrich, R. *Learning Kernel Classifiers*. MIT Press, Cambridge, 2002.

Hubert, L. and Arabie, P. Comparing partitions. *Journal of Classification*, 1985.

Jain, A K and Dubes, R C. Algorithms for clustering data, 1981.

Jain, A K, Murty, M N, and Flynn, P J. Data clustering: a review. *ACM Computing Surveys*, (31):264–323, 1999.

Johnson, Stephen. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, September 1967.

Kannan, R., Vempala, S., and Vetta, A. On clusterings: good, bad and spectral. *J. ACM*, 51 (3):497–515, 2004.

Kaufman, L. and Rousseeuw, PJ. Finding groups in data: an introduction to cluster analysis. *NY John Wiley & Sons*, 1990.

KING, B. Step-wise clustering procedures. *J. Am. Stat. Assoc*, (69):86–101, 1967.

Knuth, D. E. *The Art of Computer Programming*. Addison-Wesley, 1997.

Kruskal, J.B. and Landwehr, J.M. Icicle plots: Better displays for hierarchical clustering. *American Statistician*, 37(2):162–168, 1983. ISSN 0003-1305.

Legendre, L. and Legendre, P. Numerical ecology. *Elsevier Scientific Publishing, Amsterdam*, 1983.

Macnaughton-Smith, P., Williams, WT, Dale, MB, and Mockett, LG. Dissimilarity analysis: A new technique of hierarchical sub-division. 1964.

McQuitty, L.L. A mutual development of some typological theories and pattern-analytic methods. *Educational and Psychological Measurement*, 27(1):21, 1967. ISSN 0013-1644.

Meila, M. Comparing clusterings. Technical report, 2002.

Meila, M. Comparing clusterings by the variation of information. In *In COLT*, 2003.

Meila, Marina and Heckerman, David. An experimental comparison of model-based clustering methods. *Mach. Learn.*, 42(1/2):9–29, 2001. ISSN 0885-6125.

Meilă, Marina. Comparing clusterings—an information based distance. *J. Multivar. Anal.*, 98(5):873–895, 2007. ISSN 0047-259X.

Meilă, Marina. Comparing clusterings: an axiomatic view. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pp. 577–584, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5.

Morey, Leslie C. and Agresti, Alan. The Measurement of Classification Agreement: An Adjustment to the Rand Statistic for Chance Agreement. *Educational and Psychological Measurement*, 44(1):33–37, 1984.

Murtagh, F. Counting dendrograms: a survey. *Discrete Applied Mathematics*, 7(2):191–199, 1984. ISSN 0166-218X.

Murty, N M and Krishna, G. A hybrid clustering procedure for concentric and chain-like clusters. *Internation Journal of Computer and Information Sciences*, (10):10–6, 1981.

Narasimhan, M., Jojic, N., and Bilmes, J. Q-clustering. In *Neural Information Processing Systems*, 2005.

Rand, W M. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association 66:846850. American Statistical Association*, 1971.

Rousseeuw, P.J. A visual display for hierarchical classification. *Data Analysis and Informatics*, 4:743–748, 1986.

Rummel, R.J. *Applied factor analysis*. Northwestern Univ Pr, 1970. ISBN 0810108240.

Salton, G. and McGill, M.J. Introduction to modern information retrieval. *New York*, 1983.

Scholkopf, B., Tsuda, K., and Vert, J.-P. *Kernel Methods in Computational Biology*. MIT Press, 2004.

Scott, A.J. and Symons, M.J. On the Edwards and Cavalli-Sforza method of cluster analysis. *Biometrics*, 27:217–219, 1971.

Sibson, R. Slink: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.

Sinclair, Alistair and Jerrum, Mark. Approximate counting, uniform generation and rapidly mixing markov chains extended abstract. In *Graph-Theoretic Concepts in Computer Science*, volume 314 of *Lecture Notes in Computer Science*, pp. 134–148. 1988.

Sneath, P H A, Sokal, R R, and Eds. Numerical taxonomy, 1973.

Sokal, R R and Michener, C D. A statistical method for evaluating systematic relationships, 1958.

Wallace, D L. A method for comparing two hierarchical clusterings. *Comment, Journal of the American Statistical Association*, (78):569–576, 1983.

Ward, Joe H., Jr. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):pp. 236–244, 1963. ISSN 01621459.

Willett, P. Recent trends in hierarchic document clustering: a critical review. *Information Processing & Management*, 24(5):577–597, 1988. ISSN 0306-4573.

Wishart, D. Mode analysis: a generalization of nearest neighbor which reduces chaining effects. In *In Numerical Taxonomy*, pp. 282–311. Academic Press, 1969.

Wishart, D. K-means clustering with outlier detection, mixed variables and missing values. In *Exploratory data analysis in empirical research: proceedings of the 25th Annual Conference of the Gesellschaft fur Klassifikation eV, University of Munich, March 14-16, 2001*, pp. 216. Springer Verlag, 2003. ISBN 3540441832.

Zhang, T, Ramakrishnan, R, and Livny, M. Birch: An efficient data clustering method for very large databases. In *Proceedings of ACM SIGMOD*, pp. 103–114, 1996.